UNIVERSIDAD NACIONAL DE LUJAN

Adaptabilidad al contexto en aplicaciones web desarrolladas con continuations

por

Eloy Adonis Colell

dentro del
Departamento de Ciencias Básicas
División Estadística y Sistemas

Septiembre 2013

Declaración de Autoría

Yo, Eloy Adonis Colell, declaro que este documento titulado, 'Adaptabilidad al contexto en aplicaciones web desarrolladas con continuations' y el trabajo presentado son de mi propiedad. Y confirmo que:

- Este trabajo fue totalmente hecho mientras realizaba la candidatura para la investigación de grado en esta Universidad.
- Cuando una parte de esta tesis ha sido previamente enviada para una evaluación de nivel o alguna otra calificación a esta Universidad u otra institución, esto ha sido claramente fijado.
- Cuando he consultado el trabajo publicado de otros, esto ha sido mencionado.
- Donde he citado el trabajo de otro, la fuente siempre ha sido dada. Con la excepción de dichas citas, esta tesis es enteramente de mi propiedad.
- He reconocido todas las fuentes principales de ayuda.
- Cuando me basé en el trabajo hecho por mi mismo en conjunto con otros, he dejado exactamente claro que parte fue hecha por los otros y cual fue mi contribución a ese trabajo.

Firma: _____

Fecha: _____

"Dar el ejemplo no es la principal manera de influir sobre los demás; es la única manera."

Albert Schweitzer

UNIVERSIDAD NACIONAL DE LUJAN

Resumen

Departamento de Ciencias Básicas

Licenciado en Sistemas de Información

por Eloy Adonis Colell

En este trabajo se crea una alternativa para utilizar los sensores provistos por un navegador web desde una aplicación web desarrollada con continuations.

Diseñar dicha alternativa, contempla mezclar 2 lineas de investigación que contienen requisitos contrapuestos: la sensibilidad al contexto y las aplicaciones web desarrolladas con continuations.

Por último se presenta un ejemplo, al cual se le agrega adaptación al contexto, y se evalúa la extensibilidad y los limites de la alternativa propuesta.

Finalmente se discute el trabajo en forma global, presentando las principales diferencias con las alternativas actuales.

Agradecimientos

Primero deseo agradecer a mi Director, el Mg. Javier Bazzocco por su guía y ayuda en el desarrollo de mi tesis.

También me gustaría agradecer a mi Co-Director, el Prof. Dr. Alejandro Fernández que me motivó a aprender Smalltalk y unirme al LIFIA.

Además me gustaría agradecer a Lucas Capalbo por sus lecturas cuidadosas y comentarios constructivos que me ayudaron al mejoramiento de esta tesis.

Por último me gustaría agradecer a mis padres, por todo el ánimo y apoyo durante mis estudios.

Índice general

Índice de figuras

Índice de cuadros

Abreviaciones

API	**A**pplication **P**rogramming **I**nterface
COP	**C**ontext **O**riented **P**rogramming
CPS	**C**ontinuation-**p**assing **S**tyle
HTML	**H**yper**T**ext **M**arkup **L**anguage
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
MVC	**M**odel **V**iew **C**ontroller
QoS	**Q**uality **o**f **S**ervice
REST	**Re**presentational **S**tate **T**ransfer
URL	**U**niform **R**esource **L**ocator
UML	**U**nified **M**odeling **L**anguage
WWW	**W**orld **W**ide **W**eb

Dedicado a mis padres...

Capítulo 1

Introducción

Hace ya algún tiempo que las *páginas web* han evolucionado a lo que hoy se denomina *aplicaciones web*. Estas aplicaciones, que son accedidas y ejecutadas mediante la utilización de un *navegador web*, suelen modificar su contenido a partir de la interacción con el usuario.

El desarrollo de estas aplicaciones, a diferencia de una página web convencional, suele ser realizado mediante la utilización de algún framework[1] que simplifique la comunicación entre un *modelo de negocio* y cada uno de los posibles *usuarios*.

Existen varios frameworks que permiten simplificar las tareas de creación, actualización y mantenimiento de las aplicaciones web. La mayoría consisten en una implementación del esquema *Model-View-Controller[1, 2] (en adelante MVC)*, aunque también existe una minoría que plantea la utilización del concepto de *continuations*.

Los frameworks que implementan el esquema *MVC* contienen 3 tipos de objetos que pueden ser especializados: el *Modelo*, se encarga de la lógica de negocios; la *Vista*, se encarga de la salida del sistema; y el *Controlador*, se encarga de la entrada al sistema. Este tipo de framework suele cumplir con una arquitectura de *Transferencia del Estado Representacional*[2] *(en adelante abreviado como REST)* que garantiza que por cada interacción del usuario con el modelo de negocios, el controlador y la vista no mantendrán información de las transacciones e interacciones que realiza el usuario.

Por otra parte, existe una minoría de las aplicaciones web que son realizadas a partir del concepto de *continuations*. Estas conservan en el servidor el estado del cliente a partir de la utilización de un *Componente* como átomo de construcción. Éste cumple las

[1] Abstracción que reúne cierto código fuente genérico, permitiendo la reutilización mediante la especialización de cierta funcionalidad.

[2] Del inglés "REpresentational State Transfer".

mismas funciones que una Vista y un Controlador, primero evalúa la solicitud realizada por el cliente para poder realizar los cambios adecuados en el modelo, y luego genera el resultado que será presentado como respuesta en el navegador web.

Una *aplicación web* según el concepto de continuations, es un componente compuesto por múltiples subcomponentes (ya sean estos simples o nuevamente compuestos por otros). Además cada uno establece su flujo de trabajo, permitiendo establecer ciclos y comprobaciones mediante la utilización del lenguaje con el cual se encuentre programado el framework.

Estos frameworks que facilitan el desarrollo de las interfaces gráficas que luego serán utilizadas por los usuarios, por lo general no utilizan una parte de la información disponible en un dispositivo tecnológico (como el estado de sus sensores). De esta forma, las aplicaciones web carecen de la posibilidad de mejorar aún mas la interacción con el usuario.

En varios documentos científicos[3, 4, 5, 6, 7] se proponen alternativas para incorporar la utilización de sensores a frameworks que implementan el esquema MVC. Como contraparte, hasta el momento, no he encontrado una alternativa para realizar lo mismo con una aplicación web desarrollada con *continuations*.

1.1. Estado del arte

Dado que esta tesis pretende obtener un resultado a partir de la combinación de dos líneas de investigación: por un lado aplicaciones web que utilizan el concepto de *continuations* y por el otro la utilización de sensores como medio de interacción con el usuario (lo cual suele ser visto como un aspecto de la *sensibilidad del contexto*[8]); se prosigue con el desarrollo cronológico de cada una de ellas[3].

1.1.1. Continuations

Según la investigación histórica del surgimiento del concepto de *continuations* realizada por *Reynolds*, "Adriaan van Wijngaarden fue el primero en describir una técnica semejante a *continuations* en el año 1964, en la *IFIP Working Conference of Formal Language Description Languages*"[9, p.˜234].

De acuerdo a lo que *van Wijngaarden* describe:

[3]La palabra Contexto es utilizada en ambas áreas, en tanto se utilizará la frase *contexto de ejecución* para referirse al contexto del área relacionada con continuations. Se reservará la palabra *contexto* para lo relacionado con la sensibilidad al contexto.

> Al proveer a cada declaración de procedimiento con un parámetro formal extra - especificado como **etiqueta** - e insertar al final del cuerpo una sentencia **goto** llamando al parámetro formal. Consecuentemente, se llamará al procedimiento estableciendo una etiqueta en la posición del parámetro extra correspondiente.[10, p.˜14]

De esta manera *van Wijngaarden*, es el primero en intentar una abstracción[4] (que evita la utilización directa de *goto* y *labels*) para delegar el control de la pila de ejecución en el procedimiento al que está llamando.

Según *Reynolds*, esta abstracción fue influyendo en otros autores y evolucionando, hasta que entre 1969 y 1970 Christopher Wadsworth nombró a esta abstracción *continuations*. Actualmente, el nombre formal es *Continuation-passing style* (también abreviado como CPS).

Desde este momento en adelante es importante destacar que las implementaciones de continuations no almacenan datos del programa, sólo almacenan el *contexto de ejecución*.

En 1988, *Rees et al.* transforman a *Scheme* en el primer lenguaje en soportar de forma nativa la abstracción de continuations, denominándose a este tipo de implementaciones *first-class continuations*[11, p.˜3].

Recién en 2003, *Queinnec* sugiere la utilización de continuations en aplicaciones web como alternativa a un esquema orientado a páginas[12]. Para esto plantea el siguiente caso:

> Los navegadores brindan la funcionalidad para 'retroceder' y 'clonar'. Los clientes (por ejemplo, usuarios de los navegadores web) pueden utilizar éstas facilidades para descubrir el comportamiento del sitio con la actitud de '¿qué pasa si?'. Desde un formulario, el cliente envía información al servidor, examina el resultado (posiblemente con algo de interacción) y vuelve al formulario original si el resultado no luce prometedor. En lugar de volver atrás al formulario original, el cliente podría clonar el formulario original o *agendarlo*[5]. En todos los casos, el cliente tiene la oportunidad de volver a responder a un formulario. ¡Aún más, clonarlos permite que el cliente envíe, de forma concurrente, nuevas respuestas para formularios ya respondidos!.[12, p.˜2]

[4] Realizada en el lenguaje Algol60
[5] En inglés "bookmark it".

Queinnec, además presenta una variante que permite anular desde el servidor el proceso de 'retroceder', devolviendo al cliente el formulario correspondiente, de acuerdo con lo que el modelo de negocios establezca. Y cancelando cualquier posibilidad de volver a cargar un 'formulario' ya completo.

Esto proporciona una forma de solucionar ciertos inconvenientes (relacionados con la manipulación de transacciones) aún no resueltos de forma genérica con el esquema MVC.

En la actualidad, utilizando las nociones presentadas por *Quiennec*, Seaside es el framework mas usado por los desarrolladores dentro del conjunto de las aplicaciones web basadas en continuations. Esto le ha permitido, a su vez, liderar en el área de la investigación con el fin de brindar soportar a las nuevas tecnologías que continúan surgiendo.

1.1.2. Sensibilidad al contexto

La otra linea de investigación fundamental para el desarrollo de esta tesis es la sensibilidad al contexto.

La *computación sensible al contexto* es definida por *Schilit y Theimer* en 1994, al afirmar que "se produce cuando un *software* tiene la habilidad de descubrir y reaccionar ante cambios en el entorno en el que se encuentra un usuario móvil"[8, p.˜23]. Además agregaron que un entorno puede estar compuesto por dispositivos, personas y servicios.

Según *Dey*, *Schilit y Theimer* realizaron una definición muy acotada de contexto que sólo contempló a la localización y proximidad; por lo que en el 2001, sugiere una definición más amplia de contexto afirmando:

> Contexto es cualquier información que pueda ser usada para caracterizar la situación de una entidad. Una entidad es una persona, un lugar, o un objeto que es considerado relevante para la interacción entre un usuario y una aplicación; incluyendo al mismo usuario y/o la misma aplicación dentro de los objetos posibles.[13, p.˜3]

Como consecuencia de la definición demasiado inclusiva de *Dey*, *Efstratiou* en el 2004 propone una subclasificación describiendo:

> Las aplicaciones adaptativas sensibles al contexto, son las que modifican su comportamiento (se adaptan) de acuerdo a los cambios en el contexto de la aplicación. El término contexto es usado de acuerdo con *la definición de Dey* siendo cualquier información que pueda ser usada para caracterizar la situación de una entidad.[14, p.˜4]

De esta forma, *Efstratiou* propone que las *aplicaciones adaptativas sensibles al contexto* son un subconjunto de las *aplicaciones sensibles al contexto* y un superconjunto que contiene a las *aplicaciones adaptativas tradicionales* (ver Figura 1.1).

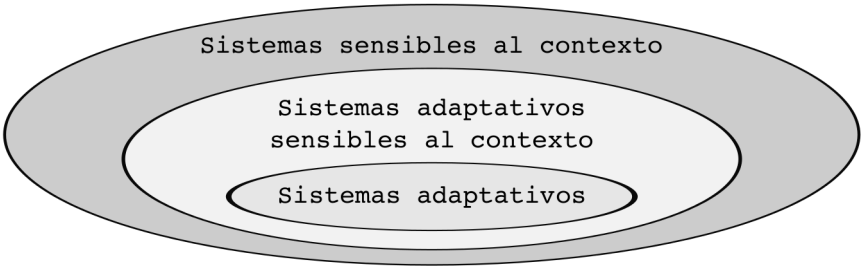

FIGURA 1.1: Sistemas adaptativos sensibles al contexto

Efstratiou, continúa detallando que "se espera que una aplicación adaptativa sensible al contexto, sea capaz de reaccionar a una variedad de *disparadores contextuales*[6]". Además, plantea que "la interacción de múltiples aplicaciones sensibles al contexto que se adaptan a distintos disparadores de entornos, al interrelacionarse podrían causar inestabilidades y comportamientos no deseados".

Por último, en su tesis explica como coordinar las diferentes aplicaciones, y como resolver los conflictos entre los distintos intereses que puede disparar cada aplicación. Además plantea como involucrar las preferencias del usuario y promover la extensibilidad.

Por otra parte, en 2008, *Costanza* plantea una postura levemente diferente a la de *Dey*. *Costanza* justifica que "*la definición de Dey* distingue entre información relevante y no relevante, lo cual es importante cuando se está modelando sistemas de adquisición y razonamiento de contextos", mientras que su definición "apunta a formas estructuradas de ver cómo se ve afectado el comportamiento del programa en función del contexto, lo cual debería ser abordado por construcciones de uso general y así ser independiente de los tipos de información de contexto de los cuales depende"[15, p.˜1].

Costanza propone un paradigma de *Programación Orientada a Contextos* (en adelante COP[7]). En este paradigma, "los programas se suelen dividir en *variaciones* (o diferenciales) de comportamiento que pueden ser activadas y combinadas en tiempo de ejecución con alcances bien definidos". El COP "se centra en construcciones de programación que habilitan el agrupamiento, el referenciamiento, y la activación/desactivación de capas de variaciones en el comportamiento". Además presenta a *ContextL* que surge como extensión al *Common Lisp Object System*[8], permitiendo asociar la definición de *clases*, *métodos* y *funciones* a distintas *capas* (o variaciones).

[6] Del inglés "Contextual triggers".
[7] Del inglés "Context-Oriented Programming".
[8] El cual es la extensión del lenguaje Common List que brinda soporte a la Programación Orientada a Objetos (o Object Oriented Programming).

Chang et al. continuando por la línea de *Dey* y *Efstratiou*, en el 2007, intuye que una aplicación web necesita adaptarse a sí misma a diferentes contextos de ejecución para garantizar cierta *calidad de servicio* (en adelante QoS[9])[16, p.˜1]. Específicamente, describe que una *aplicación web sensible al contexto* requiere:

- La habilidad de *auto-adaptarse* a los entornos específicos de su implementación.
- El potencial de *evolucionar* bajo una forma general, tanto a partir del entorno de ejecución como de la forma de uso.

Principalmente, Chang menciona que las "aplicaciones web son esencialmente distribuidas y heterogéneas", a partir de que existen 2 lugares en donde se puede realizar el procesamiento: el *cliente* y el *servidor*, y en ambos existen alternativas en cuanto a la tecnología utilizada y el entorno disponible. Como consecuencia de esto, desarrolla "la descomposición de una aplicación web en componentes que serán utilizados si es que el contexto así lo requiere". Además detalla que es suficiente con que una aplicación se adapte en el momento de *descargarse al cliente*, descartando así la necesidad de adaptación en tiempo de ejecución.

Hacia comienzos del 2009, *Kapitsaki et al.* prosigue en la investigación de sensibilidad al contexto en combinación con aplicaciones web y descarta que la adaptación al contexto deba realizarse dentro del modelo de negocios. Relacionando la sensibilidad al contexto como una característica más que puede contener un Controlador (del esquema MVC)[6].

De esta forma, *Kapitsaki et al.* propone una arquitectura en la que, mediante la composición de un grupo de servicios (o aplicaciones web) que no son sensibles al contexto, permite la adaptación al contexto dejando al controlador como responsable de la relación entre el modelo de negocios y el modelo contextual.

También en 2009, *Fortier et al.* plantea un conjunto de micro-abstracciones, que permiten lidiar con la variabilidad de los modelos sensibles al contexto en dispositivos móviles[7].

1.2. Motivación

Teniendo en cuenta el estado del arte de las aplicaciones web desarrolladas con continuations, en donde no es posible la utilización de sensores para controlar la aplicación. Y considerando las ventajas en la interacción con el usuario que brindan aquellas aplicaciones que se adaptan al contexto.

[9] Del inglés "Quality Of Service".

Se propone el desarrollo de una librería, que extienda a una implementación de Continuations, para poder utilizar en el servidor la información provista por los sensores de los distintos clientes.

Para esto se utilizará el concepto de *triggers contextuales* mencionado por *Efstratiou*, y como consecuencia se propondrá una implementación de *aplicación web adaptativa al contexto*.

En la solución propuesta, es necesario contemplar los requisitos de cada linea de investigación y tratar de encontrar el mejor compromiso entre ellas.

1.3. Contribuciones

Las contribuciones que se podrán desprender de esta tesis son:

- Principalmente, la adaptación de una aplicación web que utiliza continuations al contexto del usuario.
- En segundo término, un análisis de los patrones de diseño que se utilicen en la realización de la librería.
- Por último, una comparación de la librería propuesta con las librerías existentes para el esquema MVC.

Se esperan estas tres contribuciones antes mencionadas. En primera instancia se trata de proveer una librería de sensiblidad al contexto para las aplicaciones web que utilizan continuations.

Para demostrar la utilidad y funcionalidad de la librería se presentará un caso de ejemplo que adapta la aplicación web al contexto del dispositivo.

La segunda contribución es que la librería sea modificable y extensible. Al realizar un análisis de los *patrones de diseño*[2] utilizados se podrá entender mejor el modelo planteado.

La última contribución se enfoca en realizar una comparación de la librería propuesta y las alternativas existentes que utilizan el esquema MVC.

El alcance de esta tesis no incluirá:

- discusión alguna sobre la variante *evolutiva* de la sensibilidad al contexto, que utiliza el análisis de las interacciones del usuario para *deducir* nuevas adaptaciones al contexto;

- pruebas de rendimiento o performance, dado que la realización de estas desviarían el objetivo de esta tesis, que es demostrar la posibilidad de mezclar las dos lineas.

1.4. Esquema

El resto de la tesis se organiza de la siguiente manera: en el *Capítulo 2* se desarrollará con profundidad el concepto de continuations en el marco de las aplicaciones web. Dentro del *Capítulo 3* se introducirán los conceptos esenciales sobre aplicaciones sensibles al contexto necesarios para comprender esta tesis.

En el *Capítulo 4* se describirá la solución propuesta y se analizarán los patrones de diseño utilizados.

Un ejemplo completo será presentado en el *Capítulo 5*, en el cual se mostrará la flexibilidad de la librería en la utilización de sensores. Mientras que el *Capítulo 6* mencionará las extensiones posibles para la librería, además de los límites que ésta posee.

Por último, en las *Conclusiones*, se realizará una breve comparación con otros trabajos semejantes, se mencionarán los resultados de esta tesis, y algunas líneas de investigación que se desprenden de este trabajo.

Capítulo 2

Continuations en aplicaciones web

Como se mencionó en el capítulo anterior, el objetivo de esta tesis es incorporar la información brindada por los sensores de un dispositivo tecnológico en un framework de aplicaciones web basado en continuations.

A partir de esto, es importante destacar que las aplicaciones web desarrolladas con continuations comienzan a surgir en el año 1995. Y gran parte del aporte lo realizaron *Ladd et al.*[17] y *Quiennec* [12], quienes fueron los primeros en proponer esta variante con el fin de sobrellevar la falta de estado del protocolo *HTTP*[1].

Luego, en el 2004, *Ducasse et al.*[18] explica las ventajas de utilizar cierta implementación de continuations para aplicaciones web con respecto a otros frameworks alternativos.

A continuación se enumeran los *inconvenientes de desarrollar una aplicación web sin continuations*. Luego, como contraparte se presenta Seaside[2], una solución basada en continuations a partir de *componentes* reutilizables, ampliamente utilizada por los desarrolladores de aplicaciones web.

Por último, y considerando que Seaside va a ser el framework en donde se realizarán las pruebas pertinentes, se describirá como ha sido extendido para soportar funcionalidades como *AJAX*[3] y *Server Push*[4], que son técnicas utilizadas en gran parte de las aplicaciones web de la actualidad.

[1] Del inglés "Hypertext Transfer Protocol".

[2] Framework para desarrollar aplicaciones web orientado a objetos basado en continuations. Ver http://www.seaside.st/.

[3] Del inglés "Asynchronous Javascript And XML".

[4] Técnica que permite enviar información desde el Servidor hacia el Cliente, sin que este último la haya solicitado explícitamente.

2.1. Inconvenientes de los frameworks sin continuations

Al desarrollar aplicaciones web con frameworks que no se basan en continuations existen un conjunto de inconvenientes y/o contratiempos que deben ser resueltos. Algunos de ellos consecuencia de una escasa abstracción sobre el protocolo HTTP, y otros por restricciones planteadas por el esquema REST[19, p.˜76] existente en muchos frameworks.

Como sintetiza *Ducasse et al.*[18, p.˜234] existen dos tipos de dificultades cuando se utiliza algún framework alternativo (tal como Servlets/JSP[5], PHP[6], ASP[7], JSP[8] o Zope[9]): los relacionados con la lógica del *flujo de control* y los que conciernen al *estado* entre las interacciones del usuario.

2.1.1. Flujo de control

Al considerar el flujo de control de estos últimos frameworks, se observa que fallan en proporcionar una abstracción de alto nivel para especificar cómo son enlazadas las páginas.

La lógica del flujo de control debería estar implementada idealmente en una única porción de código, que utilice estructuras de control comunes al lenguaje de programación utilizado para especificar el modelo de negocios. Desafortunadamente, esta forma de programar es opuesta a lo que el protocolo HTTP establece (con sus mensajes de solicitud y respuesta).

Los problemas más relevantes relacionados con el flujo de control son:

Mezclar la lógica de aplicación con la lógica de los componentes. La interacción del usuario en una aplicación web puede ser vista como dos tareas que se repiten en el tiempo: la primera es *generar la página* y la segunda *procesar los datos* cuando el usuario los envíe. Estas tareas se ejecutan de forma separada y levemente conectadas.

De esta forma, la decisión de *"¿qué es lo que hago a continuación?"* está asociada a la parte de procesamiento de datos de la última página, en lugar de ser definido como un *flujo de control* a más alto nivel.

[5] http://java.sun.com/products/servlet/ y http://java.sun.com/products/jsp/
[6] http://www.php.net/
[7] http://msdn.microsoft.com/en-us/library/ms972347.aspx
[8] http://java.sun.com/products/jsp/
[9] http://www.zope.org

En los frameworks orientados a páginas cada componente debe indicar como interactúa con el resto. Así, cada página en una secuencia tendrá un enlace *estructurado de forma estática*[10] a la próxima.

Incapacidad para componer flujos de control. No todos los frameworks para desarrollar aplicaciones web proveen la capacidad de reutilizar componentes (en donde cada uno posee su propio flujo de control). Esto dificulta la posibilidad de trabajar con *múltiples flujos de control* en una misma página.

Controlar el flujo del programa. Dado que el usuario puede clonar una página e ir hacia atrás en el historial, es necesario contar con la posibilidad de *invalidar* un flujo de control luego de que una *transacción* se haya concluido y aceptado.

2.1.2. Estado

Por otra parte, una aplicación web típica tiene que tratar con tres tipos de estados: el asociado a la *interfaz gráfica con el usuario* (de cada componente), el del *modelo del dominio* (relacionado con las instancias del modelo del negocio) y el referido a la *sesión de usuario* (se encarga de mantener estados globales de la sesión, que no pertenecen al modelo de negocios ni a la interfaz gráfica con el usuario). Un ejemplo de este ultimo tipo de estado es la identificación del usuario, que suele ser común a todos los componentes.

Pero administrar el estado en el ámbito de una aplicación cliente-servidor es complejo como consecuencia del flujo de control no lineal. Esto, principalmente, es causado por la incapacidad del protocolo HTTP de mantener estados y por las capacidades de los navegadores web (clonar y/o retroceder).

Actualmente, en los frameworks orientados a páginas el estado es codificado en las respuestas al cliente. Esta solución temporal conduce a los siguientes inconvenientes:

Codificar el estado. Cuando el estado del modelo del dominio debe ser tenido en cuenta sobre una secuencia de páginas, cada página tiene que pasar la información recibida desde las páginas anteriores y trasladarla a la próxima. Esto produce que el código no sea reutilizable por su dependencia de las páginas precedentes.

Colisión de los nombres. Puede haber colisiones en los nombres de las *URL*[11] o en los campos de los formularios. El programador debe encargarse de que los identificadores sean únicos a lo largo de una página. Esto se vuelve especialmente un problema cuando se desean reutilizar los componentes.

[10] Traducción no literal de "hardcoded".
[11] Del inglés "Uniform Resource Locator".

Mezclar la lógica de presentación con la del dominio. Para codificar el estado en una página se deberá mezclar la generación de HTML[12] con la lógica de dominio usando *templates* o la concatenación de *strings*. Esto terminará produciendo código fuente difícil de leer.

2.2. El *Componente* como bloque de construcción

A partir de los inconvenientes planteados con el flujo de control y la administración de estados, existen frameworks como Seaside en los que se propone la construcción de una página a partir de *componentes*[13]. Cada componente se encarga de definir la interfaz de usuario y el flujo de control de una parte de la aplicación. A diferencia de los frameworks orientados a páginas la instancia de un componente a menudo perdura a lo largo de la sesión del usuario.

En cada solicitud el framework le permite a cada componente evaluar sus *callbacks* y *renderizar* su estado actual en un *stream de respuesta*.

Las características más importantes de esta aproximación son: el *renderizado*, la *composición de componentes* y los *action callbacks*.

Renderizado. Cada componente que es visible en una aplicación web tiene un *método de enganche*[14], que permite definir su aspecto visual. Este método es llamado con un *stream de respuesta* como parámetro, permitiendo especificar las características gráficas y de comportamiento de cada componente con el mismo lenguaje en el cual está implementado el modelo de negocios.

Composición de componentes. Para la construcción de una aplicación web un componente puede contener a muchos otros. De esta forma una "página" sería definida como un componente compuesto. Por otra parte existe un componente específico denominado *Task*, que se encarga solamente de definir un flujo de control, y así prescindir del *renderizado*.

Action Callbacks. Los componentes pueden reaccionar a las acciones realizadas por el usuario a través de los *action callbacks*. Estos son definidos sobre los *botones*, *enlaces* y por cada campo de un formulario mediante la utilización de un *bloque de código o functor* al momento de describir el comportamiento de un componente.

[12] Del inglés "Hypertext Markup Language".
[13] Seaside lo implementa en la clase *WAComponent*.
[14] Del inglés "callback".

El *action callback* de un campo del formulario contiene un argumento que representará al valor del campo en cuestión. Mientras que para los enlaces y botones el bloque no contiene ningún argumento, en estos se define el flujo de control del componente o de la aplicación.

Dado que cada componente define su propio flujo de control independiente de los otros mostrados en la misma página, un componente o aplicación puede tener múltiples flujos de control.

Cada vez que el usuario utilice un *enlace* o un *botón* en un componente realizará un paso hacia adelante en su flujo de control, mientras el resto de los flujos de control permanecen en el mismo estado.

Para facilitar la creación y administración de múltiples flujos de control los frameworks basados en *componentes* proporcionan los métodos **call:** y **answer:**.

Call. Mediante este método un componente (por ejemplo A) puede pasarle el control a cualquier otro componente (en este caso B). Durante este tiempo el componente original (A) es temporalmente reemplazado por el otro componente (B). Esto es posible mediante el envío del mensaje **call:** al componente A y enviándole como parámetro el componente B.

Answer. Mediante este método un componente (por ejemplo el B) puede devolver el control a aquel componente desde el cual ha sido llamado (en este caso el A). Cada componente eventualmente devolverá el control de la ejecución en algún punto, e inclusive puede retornar un objeto que modele la respuesta. Esto es posible mediante el envío del mensaje *answer:* al componente B y enviándole como parámetro el objeto que modela la respuesta.

Aunque desde la perspectiva del desarrollador el código es lineal, existen algunas cuestiones relacionadas con el estado de la aplicación web que deben ser explicadas.

Seaside permite la *clonación* de una página y *volver* al estado anterior. De esta forma es necesario que exista cierto mecanismo que se encargue de mantener el estado de la *interfaz gráfica* y del *modelo del dominio*, ambos mencionados en la sección Estado (2.1.2).

2.2.1. Backtracking

Dado que cada componente tiene su propio estado, el cual es guardado en las variables de instancia (por ejemplo un componente que muestra un listado de elementos de forma

paginada recordará el numero de la página actual). Cuando el usuario *vuelve* hacia atrás con el navegador web, las variables de instancia del componente pueden no representar el estado que el usuario visualiza.

Una forma de mantener el estado de los componentes en sincronía con lo que visualiza el navegador web es mediante la utilización de continuations. Para esto, Seaside ofrece un mecanismo para seleccionar qué variables de instancia deben ser *rastreadas*[15]. Después de cada respuesta enviada al cliente, el framework guarda una copia exacta de los objetos seleccionados mediante la creación de un *registro de activación*.

La sesión del usuario almacena el registro como una variable temporal dentro del proceso que envía la *respuesta* al cliente y que luego recibirá la próxima *solicitud*. El registro se vuelve persistente hasta un momento en el futuro en el cual el flujo de control requiera retomar la ejecución de este componente almacenado.

Al momento de restablecer el *contexto de ejecución* y antes de procesar la solicitud del cliente, el registro restablece los objetos registrados y la *continuation* es procesada.

Esto asegura que cuando se procesa una solicitud, los valores son los mismos que cuando la respuesta previa fue creada.

2.2.2. Transacciones

En ciertas aplicaciones complejas suele ser necesario interceder para prevenir que el usuario retroceda a la página anterior. Seaside implementa el método "*isolate: aBlock-Closure*" para controlar el flujo de control. Este método recibe un *bloque de código* o *functor* como argumento, el cual es tratado como una transacción.

La transacción le garantiza al usuario poder ir hacia adelante y hacia atrás, siempre y cuando permanezca dentro del *bloque transaccional*. Sin embargo, una vez terminada la transacción el usuario no podrá volver hacia atrás, dado que la transacción ha sido dada por concluida.

2.3. Combinación con las nuevas tecnologías

Al considerar el requisito de extender una aplicación web basada en continuations, es necesario corroborar como se han realizado otras extensiones al framework presentado.

[15] Del inglés "backtracking".

Para esto, a continuación se repasará como otras librerías de terceros han incorporado dos protocolo de comunicación como *AJAX*[16] y *Server Push* al framework Seaside.

2.3.1. Asynchronous Javascript And XML

El uso masivo de las aplicaciones web comienza a evidenciar que las interacciones del usuario se realizan sobre ciertas partes de una página, mientras que otras partes permanecen inalteradas.

Como consecuencia se comienza a trabajar en una abstracción que permita desacoplar la interacción del usuario de la carga de la página, con el fin que puedan realizarse solicitudes de forma asíncronas y manipular las respuestas para que puedan ser incluidas en la porción de página que debe ser actualizada.

Esta abstracción es denominada *AJAX* y es totalmente compatible con el protocolo HTTP, dado que mantiene el requisito que una *respuesta* del servidor solo será dada a partir de una *solicitud* del cliente[17].

Para soportar *AJAX* de una forma transparente al desarrollador, Seaside agrega la clase *SUComponent* que es una especialización del *WAComponent*[18] (ver Figura 2.1).

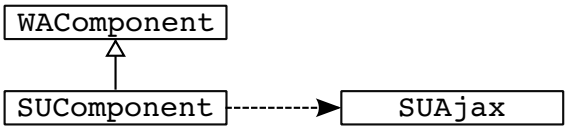

FIGURA 2.1: Extensión de Seaside para soportar AJAX

El componente *SUComponent* es una implementación del patrón *Abstract Factory*[2] que permite la creación de objetos *SUAjax*.

Cada instancia de *SUAjax* representa una posible solicitud asíncrona que puede realizar el cliente. Luego, en el momento del *renderizado*, el framework podrá procesar la respuesta que posteriormente será enviada al cliente.

2.3.2. Server Push

La utilización de AJAX dio lugar a un nuevo conjunto de aplicaciones web que mejoran la interacción e intercomunicación entre distintos usuarios.

[16] Es la sigla de "Asynchronous Javascript And XML".
[17] Detallado en la *RFC2616* (http://www.ietf.org/rfc/rfc2616.txt).
[18] Componente base del framework Seaside.

Para esto un navegador web debe preguntar al servidor de forma periódica si existen actualizaciones para cierto usuario. Cuanto mas seguido pregunta, más interactiva se vuelve la aplicación. Aunque existen ciertas aplicaciones web en donde el cliente consulta muchas veces y las actualizaciones son por momentos pocas y por momentos muchas.

Esto planteó otra situación, y es que los navegadores solo pueden obtener las actualizaciones de los otros usuarios a partir de solicitarlas al servidor, como consecuencia del ciclo solicitud/respuesta del protocolo HTTP.

Para mejorar esto, *Server Push* utiliza conexiones *HTTP* de larga duración para reducir la *latencia* de comunicación entre el *servidor web* y el *navegador*. Esto implica una *optimización* con respecto a AJAX, al permitir que un servidor inyecte información en un cliente en particular.

Desde el punto de vista del cliente, en lugar de solicitar reiteradamente al servidor por la existencia de alguna actualización, el navegador deja una conexión abierta[19]. De esta forma el servidor cuenta siempre con un canal para enviar información, teniendo la capacidad de establecer el momento en el que el cliente recibirá la información.

Para implementar *Server Push*, *Seaside* agrega la clase *Meteoroid* que es una especialización del *WAComponent* (ver Figura 2.2).

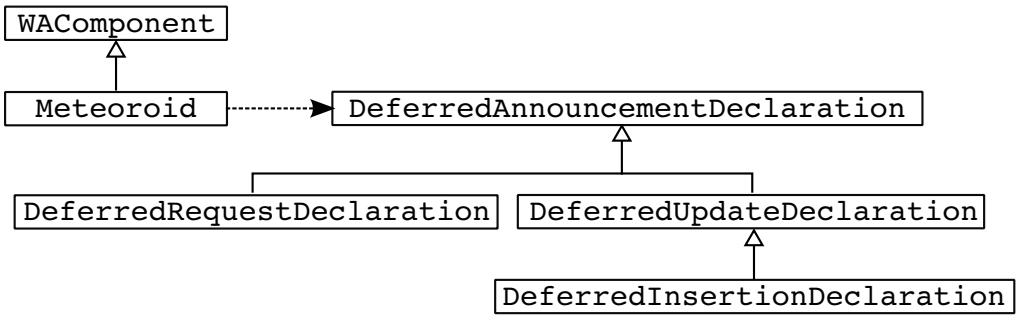

FIGURA 2.2: Extensión de Seaside para soportar Server Push

El componente *Meteoroid* permite actualizar el contenido del cliente web mediante tres acciones: una *solicitud*, la *inserción* y la *actualización*.

La *solicitud* posibilita que el servidor obligue al cliente a realizar una *solicitud asíncrona mediante AJAX*.

Por otra parte, mientras la *inserción* siempre agrega el contenido al final de una *etiqueta HTML*, la *actualización* reemplaza el contenido completo.

[19] Conceptualmente se considera que es una conexión abierta, aunque en realidad se suele utilizar un *timeout* elevado que eventualmente podrá cerrar la conexión.

Capítulo 3

Sensibilidad al contexto

Luego de describir a Seaside como referente para desarrollar aplicaciones web basadas en continuations, es necesario explicar que alternativas existen para procesar la información provista por un conjunto de sensores.

Una forma de organizar la utilización de sensores es mediante el concepto de *sensibilidad al contexto*.

Para poder definir la sensibilidad al contexto es necesario repasar que significa *contexto*, *entorno* y *adaptación*; para luego poder definir la *sensibilidad al contexto*.

La primera definición de *contexto*, que es la más adoptada, es la que aporta *Dey* en 2001, que estipula:

> Contexto es cualquier información que pueda ser usada para caracterizar la situación de una entidad.[13, p.˜3]

Por su parte, *Dourish* intenta destacar una característica dinámica al intentar abstraer al contexto y en 2004 afirma:

> Contexto es un concepto de lo que se mantiene al margen, y se disuelve cuando uno intenta definirlo.[20]

De esta forma *Dourish* destaca lo complejo que es definir un modelo para representar al contexto, dado que cierta información del contexto puede transformarse en información del modelo y viceversa. Además, presenta un modelo contextual basado en interacciones, que se enfoca en responder "¿cómo y por qué las personas mantienen un entendimiento

mutuo del contexto en el curso de las interacciones de sus acciones?"; evitando así la definición de un contexto *estable* y *delineable* en tiempo de implementación[20, p.˜5].

Por otra parte, el *entorno* es la caracterización de las circunstancias de una situación en particular combinado con la descripción de la adaptación a realizarse para mejorar la interacción con el usuario de ese contexto. Es el punto de conexión entre un modelo de negocios y el modelo del contexto, que se encarga de describir ante qué circunstancias se realiza cierta *adaptación*.

Luego, la *adaptación* consiste en realizar un conjunto de acciones de forma automática para simplificar la interacción de una entidad con uno o varios sistemas. Aunque estas acciones sólo afectarán al modelo de negocio, se debe tener en cuenta que cierta información del contexto puede transformarse en información del modelo de negocio y viceversa.

A partir de estos conceptos, se considera que la *sensibilidad al contexto* es una característica de los sistemas que simplifica la interacción con un usuario (o entidad) a partir del *entendimiento* de la situación del mismo (o la misma). Ese *entendimiento* está determinado por la capacidad de un sistema de identificar un *entorno* y luego proporcionar una *adaptación*.

A continuación, dado que la sensibilidad al contexto se encuentra estrechamente relacionada con los sistemas adaptativos, se definen a las *aplicaciones adaptativas sensibles al contexto* de *Efstratiou*[14] para delinear el alcance de esta tesis. Luego, se introducen los conflictos que existen cuando se procede a combinar la sensibilidad al contexto con un framework de aplicaciones web basado en continuations.

3.1. Aplicaciones adaptativas sensibles al contexto

Los sistemas adaptativos[21, 22, 23] están basados en la teoría del *control mediante la retroalimentación*[1] (ver Figura 3.1).

El *controlador*[2] se encarga de mantener el valor de referencia de una variable de control, mientras reduce la sensibilidad del sistema a posibles perturbaciones. El controlador interactúa con el sistema a través de *monitores* y *actuadores*.

Un monitor mide la variable controlada, y es la fuente de retroalimentación. La salida del controlador causa que el actuador realice la adaptación al comportamiento del sistema en respuesta a las perturbaciones (o cambios en el entorno).

[1] Utilizado por primera vez en la ingeniería electrónica.

[2] Este es el *controlador* de la adaptación y no se encuentra directamente relacionado con el controlador del MVC.

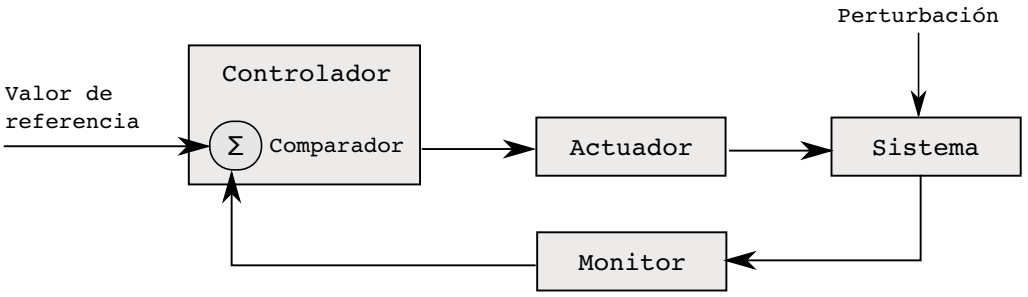

FIGURA 3.1: Sistema de control mediante la retroalimentación

La abstracción de este sistema puede sintetizarse en un *ciclo de adaptación básico* (ver Figura 3.2) que incluye los siguientes elementos funcionales:

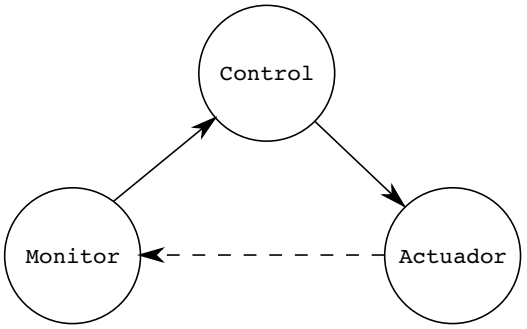

FIGURA 3.2: Ciclo de adaptación básico

Monitor: El primer elemento realiza el monitoreo de una fuente de información específica que es *interesante* para el mecanismo de adaptación. Esta fuente de información puede estar determinada por la disponibilidad de un recurso específico o un *disparador contextual* que notifique la ocurrencia de un evento.

Controlador: El segundo elemento es el mecanismo de control que toma las decisiones necesarias para adaptarse al entorno. Esta decisión se basa en la información recibida por el monitor.

Actuador: El tercer elemento es el encargado de realizar la adaptación mediante la ejecución de acciones correctivas, a medida que el controlador lo disponga. La conexión entre el actuador y el recurso que está siendo monitoreado no necesariamente existe en todos los sistemas.

Efstratiou[14] destaca que una *aplicación adaptativa* adquiere la característica de ser *sensible al contexto* cuando la información a ser monitoreada, en lugar de provenir de un recurso específico, proviene del *contexto externo* de la aplicación (cualquier porción de información que actualmente no pertenezca al *modelo de negocio*).

3.2. Conflictos entre la sensibilidad al contexto y las aplicaciones web desarrolladas con continuations

Al comenzar a combinar las aplicaciones web basadas en continuations con técnicas utilizadas por las aplicaciones adaptativas sensibles al contexto surgen dos situaciones de conflicto: la *privacidad* y las *transacciones con adaptación al contexto.*

3.2.1. Privacidad

Las *aplicaciones web* se ejecutan encapsuladas en el navegador web para proveer de cierta seguridad y privacidad al usuario. Se intenta separar su información privada y el conjunto de dispositivos conectados a una computdadora de las inseguridades que provee una conexión a Internet (o intranet).

Por otra parte las *aplicaciones adaptativas sensibles al contexto* necesitan el acceso a la información privada del usuario y a un conjunto de sus sensores, con el fin de identificar el contexto (o entorno) de forma automática y promocionar adaptaciones que simplifiquen la actividad cotidiana del usuario.

Ante esta situación conflictiva, en donde una de las partes restringe el acceso al sistema y la otra requiere todo lo contrario para funcionar, es importante presentar una solución intermedia que brinde la información suficiente para la adaptación, siempre y cuando el usuario lo desee. En el mas restrictivo de los escenarios, el usuario terminará utilizando una aplicación web convencional en donde no exista sensibilidad al contexto.

3.2.2. Transacciones

Las aplicaciones web basadas en continuations suelen contener flujos de control con soporte para transacciones. Una transacción garantiza que un conjunto de acciones sean ejecutadas de forma consecutiva, y ante algún inconveniente existe un mecanismo de *restauración* para dejar el sistema en el estado previo.

Las transacciones son utilizadas por aplicaciones con ejecuciones concurrentes, en donde se desea evitar la corrupción o inconsistencia de los datos al ser persistidos.

Como contraparte, en las aplicaciones adaptativas, las acciones son ejecutadas a partir de una necesidad de adaptación de un sistema a un contexto dado. Una acción siempre depende del entorno que motivó la adaptación.

En las aplicaciones adaptativas la persistencia de la información suele ser realizada de forma acumulativa, de tal manera que se memorice el progreso de la adaptación a través del tiempo. A partir de estos datos, un sistema evolutivo podría reconocer nuevos patrones de comportamiento de un usuario, y proveer nuevas adaptaciones.

Esto produce que no se necesiten modificar valores previos en los sistemas adaptativos, descartando en gran medida la necesidad de transacciones.

El conflicto entre ambas partes ocurre cuando una transacción de un sistema basado en continuations falla y se desea restaurar el sistema a una versión previa. En los sistemas adaptativos sensibles al contexto la única interpretación que admitiría restablecer los registros a un estado previo es aquella que se produciría si se pudiese retroceder el tiempo y sus consecuentes estados.

Dado que esto es improbable, resulta necesario contemplar una alternativa que permita mezclar las transacciones de continuations con la "progresividad" de la sensibilidad al contexto.

Para evitar esta situación, se descarta el análisis de cualquier variante *evolutiva* de sensibilidad al contexto en donde se requiere de este tipo de persistencia y se opta por integrar la adaptación al contexto sobre el framework que administra las continuations.

Esto significa que Seaside, administrará las transacciones (encargandose del estado correspondiente) y luego la sensibilidad al contexto por sobre Seaside reconocerá entornos y le notificará al framework para que ejecute las acciones de adaptación correspondiente.

Estas ejecuciones asociadas a ciertos entornos, podrán deshacerse fácilmente al no persistirse la información perteneciente a la sensibilidad al contexto. Solo se persistirán las consecuencias de las adaptaciones en el modelo de negocio de la aplicación transaccional.

Capítulo 4

Diseño

Al considerar los objetivos planteados en la introducción, que promueven la incorporación de la sensibilidad al contexto a las aplicaciones web desarrolladas con continuations, y los conflictos planteados en el capítulo anterior; surge la necesidad de diseñar una abstracción que busque una buena relación entre la sensibilidad al contexto y la privacidad del usuario.

La alternativa propuesta en esta tesis consiste en la creación de una librería que permita conocer el estado de los sensores del cliente desde el servidor, manteniendo la capacidad *transaccional* de las aplicaciones web basadas en continuations, a la vez que se intenta mantener la *privacidad del usuario* proporcionada por los navegadores web convencionales.

A continuación se detallan las decisiones de diseño tomadas comenzando desde el navegador web y culminando en el servidor. Primero se analizará la *privacidad del usuario* en el cliente. Luego se prosigue con el *mecanismo de suscripción/notificación en el cliente*, para posteriormente explicar el mismo *mecanísmo visto desde el servidor*.

Por último se describe el proceso necesario para *brindar soporte a un nuevo sensor*, y se presenta el *modelo final*.

4.1. Privacidad del usuario

La privacidad del usuario de una aplicación web se encuentra determinada tanto por los navegadores web, como por la forma en que fue desarrollada esa aplicación.

La privacidad proporcionada por los navegadores web convencionales, consiste en aislar la ejecución de la aplicación web de una gran cantidad de información disponible en un dispositivo (como archivos, otros dispositivos conectados, sensores, etc).

Por otro lado, dado que la sensibilidad al contexto requiere información específica del usuario, es necesario acceder a cierta información protegida por la mayoría de los navegadores web (valores de los sensores que permitan reconocer el entorno del dispositivo, y a su vez parte del entorno del usuario).

Una forma de permitir el acceso a esta información desde las aplicaciones web se logra al proporcionar una API[1] que acceda a los valores de los sensores. Además, el navegador web, deberá proporcionar una interfaz para que el usuario pueda establecer a que información puede acceder la aplicación web.

Por ejemplo, el navegador web utilizado para la realización de esta tesis[2](basado en Phonegap) al presionar la tecla menú (en un dispositivo móvil) muestra una lista de sensores, desde donde se podrá seleccionar cuales serán compartidos mediante la API (Ver Figura 4.1).

FIGURA 4.1: Captura de pantalla del navegador web Phonegap

Luego, dentro de las aplicaciones web, las interfaces (disponible desde Javascript) dependen de la velocidad con que se actualiza la información del sensor.

[1]Del inglés "Application Programming Interface".
[2]Construido como consecuencia de la escasez de este tipo de navegadores web.

Aquellos sensores que recuperan información altamente cambiante (como el acelerómetro, el compás y la posición) proporcionan una interfaz para configurar un *timer*[3] mediante una función que requiere de 3 parámetros (ver Figura 4.2, linea 4).

```
1 succes = function (listened) { };
2 fail = function (error) { };
3 options = { frequency: 1000 };
4 timerId = navigator.geolocation.watchPosition( sucess , fail , options );
```

FIGURA 4.2: Ejemplo de API en el lenguaje javascript para los sensores altamente cambiantes

Los 2 primeros parmámetros son *callbacks* (ver lineas 1 y 2) y el último un diccionario con *opciones* de configuración (ver linea 3). El primer callback se ejecuta cuando se ha podido obtener la información de forma correcta, y el otro en caso de que el sensor se haya encontrado con algún inconveniente.

Por otra parte, aquellos sensores que recuperan información menos dinámica (como la batería, la red a la que se encuentra conectado, etc.) suelen proveer una función para solicitar el valor actual.

Estas herramientas proporcionadas por el navegador web para que el usuario pueda seleccionar que parte de su contexto desea compartir son una parte de la solución al problema de la privacidad. La otra parte de la solución se encuentra en la forma que se encuentra desarrollada la aplicación web.

Si se tiene en cuenta la privacidad del usuario al momento de diseñar una aplicación web, existen ciertas decisiones de diseño que pueden ser tomadas.

Un aspecto importante de la privacidad del usuario, es reducir al mínimo la transferencia de información de los sensores hacia los servidores, comunicando solo aquella información que sirva para tomar una decisión en el servidor.

Este tipo de optimización, puede ser realizada al utilizar el patrón de diseño *Publish/Subscriber* descripto por *Birman et al.*[24], que permite registrarse a un entorno de interés, en lugar de requerir que el cliente envíe el estado de sus sensores de forma continua.

De esta forma, el servidor deberá registrarse a un entorno determinado (mediante algún mecanismo de *Server Push*) y el cliente realizará la notificación pertinente cuando el entorno sea compatible con el contexto del usuario.

[3]Tarea que se ejecuta de forma reiterativa dado cierto intervalo de tiempo.

4.2. Mecanismo de suscripción a un entorno

Para realizar una suscripción a un entorno, es necesario diseñar una estructura que detalle las características que permiten describirlo.

Hay entornos que dependen de la información de un solo sensor, mientras que existirán otros que dependen de una combinación de varios. Inclusive, sería deseable, que se puedan definir *sub-entornos* que luego puedan ser reutilizados en diferentes entornos mas complejos.

Para obtener esta *granularidad* en la definición del entorno, se utiliza el patrón de diseño *Composite* descripto por *Gamma et al.*[2], mediante las clases *Condition*, *SimpleCondition* y *ComplexCondition* (ver Figura 4.3).

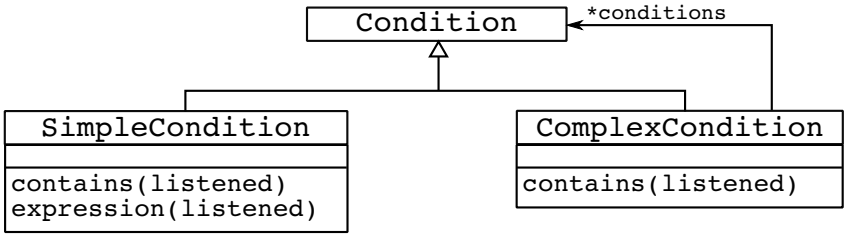

FIGURA 4.3: Implementación del patrón Composite mediante la clase Condition

La clase *SimpleCondition* representa al estado deseado de un sensor en un momento determinado. Es la menor granularidad posible dentro de la definición del entorno, pudiendo evaluarse si el estado de un sensor cumple con su requisito mediante el método *contains*.

Por otra parte, la clase *ComplexCondition* permite la caracterización de entornos a partir de la utilización de varios sensores. Cuando se evalúa una condición compleja (utilizando el método *contains*) se tiene en cuenta que todas las condiciones que la componen se cumplan. En caso de no cumplirse alguna de las condiciones, la condición compleja tampoco se cumplirá.

Por último, para homogeneizar las distintas interfaces (o API) con las que se puede acceder a la información de un sensor (mediante el patrón de diseño *Adapter*[2]), se crea la clase *Listener* que abstrae a las *SimpleCondition* de la API específica del Phonegap (ver Figura 4.4).

Dentro de una instancia de la clase *Listener*, el método *startListening* se encargará de inicializar un *timer* (o, dependiendo del caso, de configurar el *timer* perteneciente a la API de Phonegap) que será el encargado de leer el valor. En caso de éxito se llamará al método *success* y en caso de error al método *fail*.

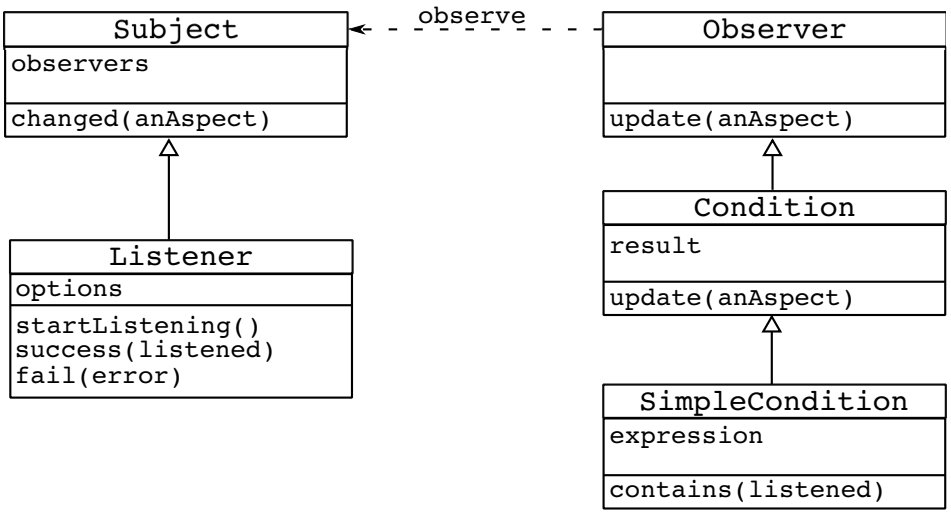

FIGURA 4.4: Abstracción de los sensores mediante la clase Listener

Cada instancia de *SimpleCondition* observará (mediante el patrón de diseño *Observer* presentado por *Gamma et al.*[2]) a una instancia de la clase *Listener*. De esta forma, cuando un *Listener* conozca una nueva lectura de su sensor, las *SimpleCondition* que lo observan podrán actuar al respecto.

Este proceso se encuentra detallado dentro del método *success* de la clase *Listener* en donde se notifica a cada *SimpleCondition* que se encuentra observando, utilizando el método *update*. Dentro de *update*, cada *SimpleCondition* evaluará si contiene (utilizando el método *contains*) al valor informado por el sensor.

Una vez definidas las clases que permitirán la descripción y detección del entorno, es necesario detallar el mecanismo de notificación al servidor.

4.3. Notificación al servidor de la ocurrencia de un entorno

Teniendo en cuenta que el servidor se suscribió a un entorno detallado (descrípto por un conjunto de condiciones), cuando el cliente realice la notificación no requerirá el envío de información sobre el entorno.

El cliente solo informará cual es el *callback de continuations* que el servidor debe ejecutar mediante una solicitud *AJAX*, la cual ha sido administrada por el servidor al momento de registrarse al cliente. El cliente guarda dicha solicitud de forma *serializada* en el atributo *callback* de la clase *Trigger*.

Luego, una instancia de la clase *Trigger* se relaciona con una instancia de la clase *Condition*, quedando asociado la descripción de un entorno determinado con una notificación al servidor (ver Figura 4.5).

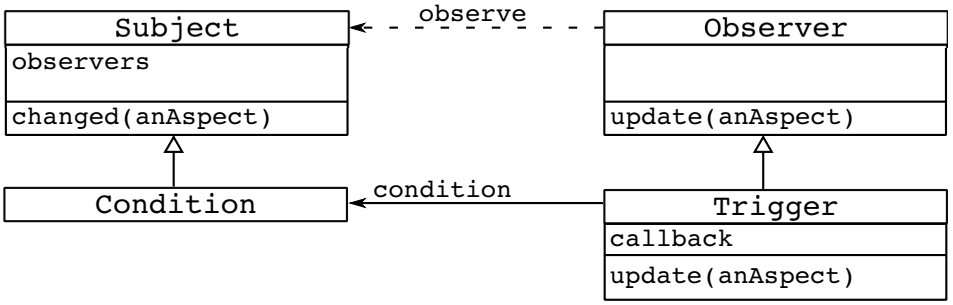

FIGURA 4.5: Implementación de la clase Trigger como parte escencial del patrón Public/Subscriber

Notar que la clase *Condition* además de ser un *observador* es un *subjeto*, a partir de esto podrá ser observada por una instancia de la clase *Trigger* o por una instancia de *ComplexCondition*.

De esta forma, cuando la clase *Listener* recupera un nuevo valor del sensor, se lo notifica a todas las *SimpleCondition* que lo observan. Éstas, dependiendo del caso particular, notificarán a las *ComplexCondition* (que podrán notificar a otras *ComplexCondition*), hasta llegar a todos los *Trigger* involucrados (ver Figura 4.6).

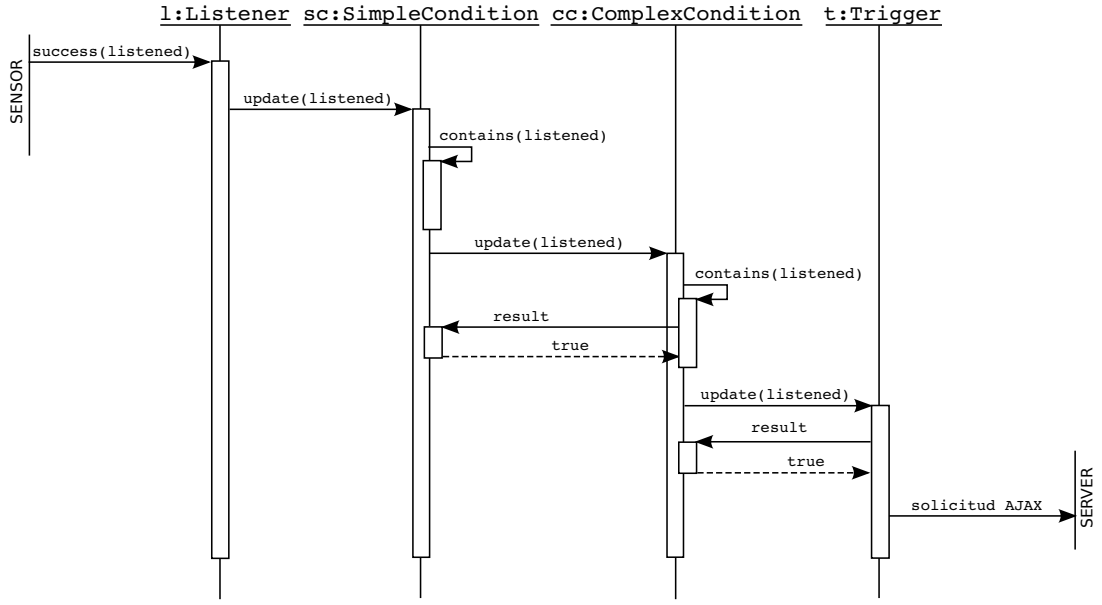

FIGURA 4.6: Diagrama de secuencia UML del proceso de notificación

Cada vez que un *Trigger* es notificado de una actualización en su atributo *condition*, verifica si la condición describe al entorno actual. En caso positivo, enviará la solicitud AJAX al servidor, informándole cual es el *callback* que debe ejecutar.

4.3.1. Optimización de la propagación

Cada vez que el *listener* notifica a las condiciones que lo observan, estas propagarán la notificación hasta llegar a los respectivos *triggers* asociados (ver Figura 4.7).

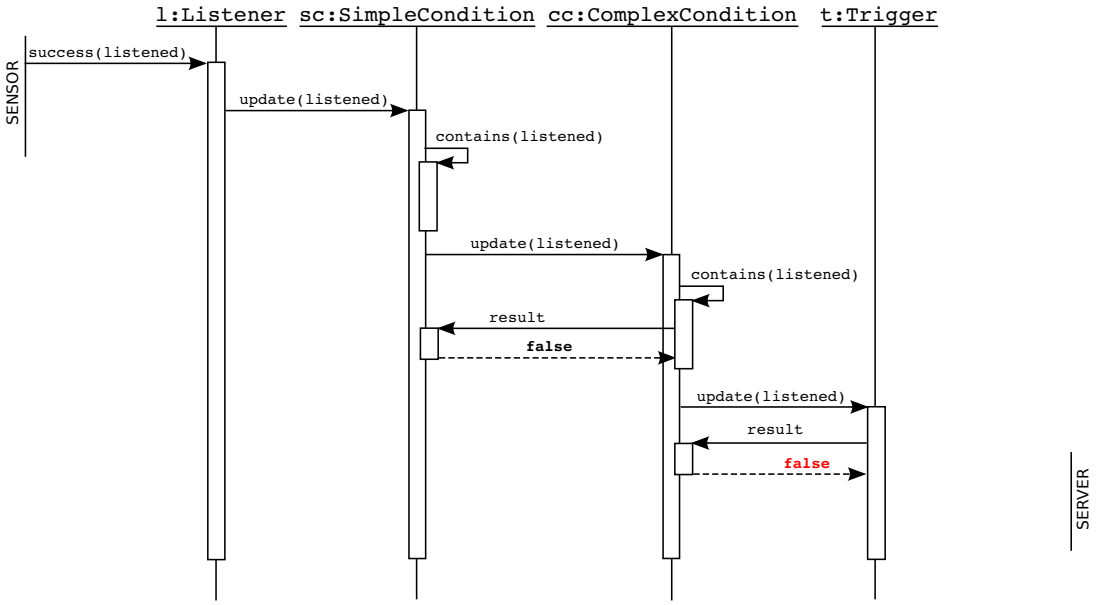

FIGURA 4.7: Diagrama de secuencia UML de la propagación hasta llegar al Trigger

El trigger es el que verifica si se debe notificar al servidor mediante la solicitud AJAX. Una forma de optmizar esto es frenar la propagación de la notificación en las condiciones previas, antes de que llegue al trigger, si es que la situación lo permite.

Para realizar esto se almacena el resultado de cada evaluación para cada condición. Esto permite verificar si la condición cambió de resultado con respecto a la evaluación anterior, permitiendo interrumpir la propagación en aquellos casos que lo requieran (ver Cuadro 4.1).

anterior/actual	falso	verdadero
falso	para	continua
verdadero	continua	continua

CUADRO 4.1: Propagación de la notificación en las condiciones

Como se observa en la tabla cuando el resultado de la evaluación de una condición es *falso*, y su resultado anterior fue *falso* no es necesario notificar un cambio a los observadores de esta condición.

4.4. Mecanismo de cancelación de una suscripción en el cliente

Al considerar la interacción de un modelo con el contexto de sus usuarios, es posible contemplar que en algún momento el servidor deseará cancelar alguna de sus suscripciones.

Para poder realizar la cancelación es necesario que el servidor mantenga identificados todos los objetos enviados al cliente (triggers, condiciones simples y complejas). De esta forma puede especificarle al cliente cual de todos esos objetos quiere eliminar (o cancelar). Para lograrlo es necesario modificar la jerarquía de clases antes presentada, haciendo que la clase *Condition* y la clase *Trigger* extiendan a la clase *ProxyObject* (ver Figura 4.8).

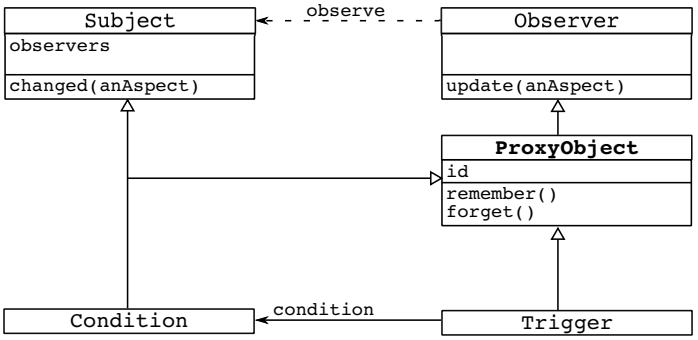

FIGURA 4.8: Diagrama de clases UML con la clase ProxyObject

Además la clase *Contidion* continuará extendiendo a la clase *Subject* para poder notificar a sus observadores en caso de ser necesario.

Por último es necesario subrayar que la clase *ProxyObject* es la que permite reutilizar los *sub-entornos* definidos utilizando instancias de la clase *ComplexCondition*.

4.5. Suscripción y desuscripción desde la perspectiva del servidor

Teniendo en cuenta el esquema de notificaciones del cliente basados en el *ProxyObject*, es necesario ampliar como el servidor construye la información para que luego pueda ser interpretada por el navegador web.

El proceso comienza cuando a una instancia de la clase *Listener* (que reside en el servidor), asociada a un cliente en particular, se le envía el mensaje *#remember:* con una instancia de *Trigger* como parámetro (ver Figura 4.9).

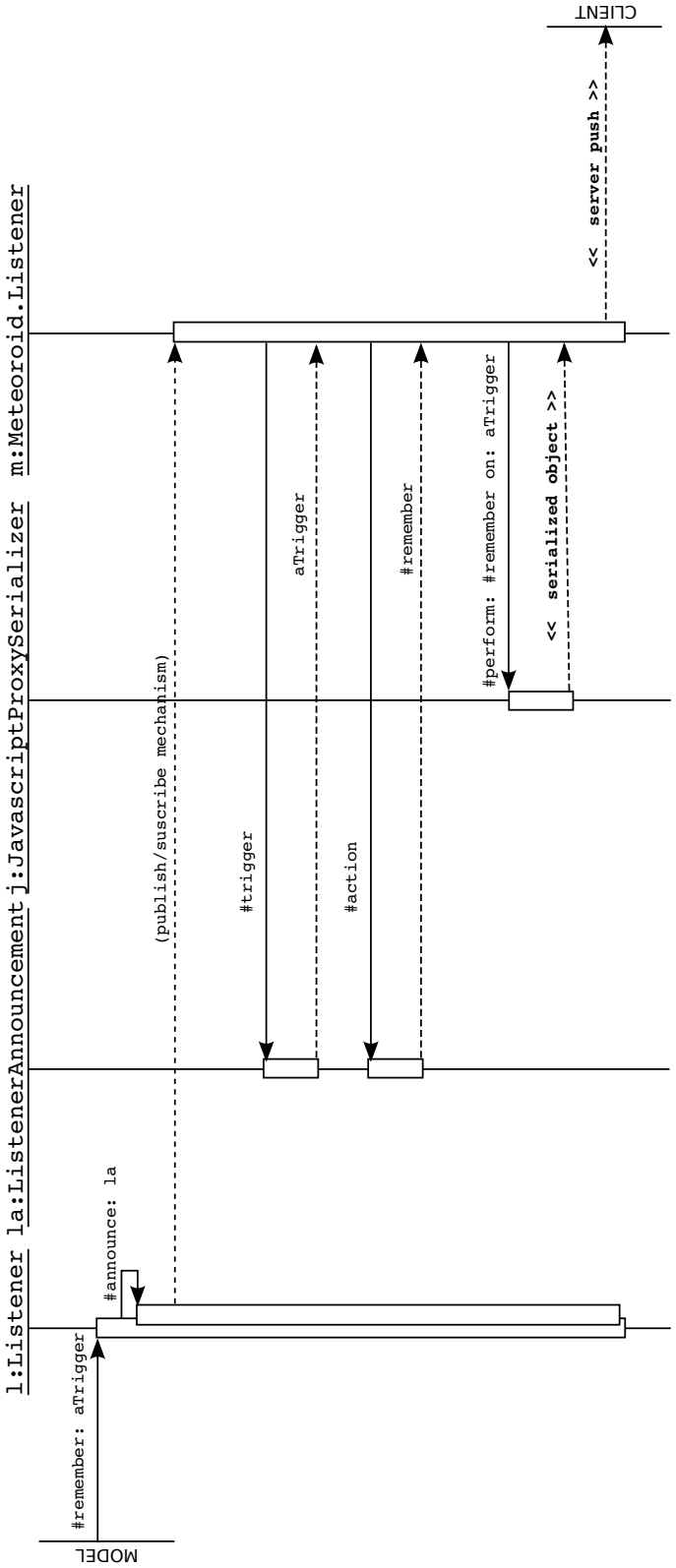

FIGURA 4.9: Diagrama de secuencia UML de la serialización de objetos en el servidor

La instancia de la clase *Listener* es observada por la clase *Meteoroid.Listener*, que es un componente del framework *Meteoroid*. Al utilizar el patrón de diseño *Publish/Suscriber*,

el componente de meteoroid es notificado cada vez que al Listener se le indica *recordar* o *olvidar* algún trigger.

Luego, al recibir una notificación la instancia de *Meteoroid.Listener* le delega a una instancia de la clase *JavascriptProxySerializer* para que recorra al objeto *Trigger* con el fin de transformarlo en un código javascript adecuado para el navegador web.

El código javascript resultante describirá a las *condiciones* que debe cumplir el entorno para ejecutar ese *Trigger* y almacenará un *callback de continuations* codificado como una solicitud de AJAX.

Por último, cuando el serializador devuelve el código javascript, el componente lo inyecta en el cliente (utilizando la técnica conocida como Server Push).

4.6. Implementación de un sensor

En las secciones anteriores se presentó el mecanismo de notificación de entornos en forma general. Este mecanismo le permite al servidor (basado en continuations) interactuar con un cliente dentro del dominio de la adaptabilidad al contexto. Para concluir con el entendimiento de lo explicado se prosigue con el desarrollo de un caso particular, como es el caso del *sensor* de *aceleración*.

Teniendo en cuenta que la aceleración proviene de un sensor altamente cambiante, el navegador web basado en *Phonegap* provee en su API un método para configurar un *timer*.

Para *adaptar* esta API, se crea la clase *AccelerationListener* (ver Figura 4.10) que extiende a la clase *Listener* e integra al sensor al mecanismo de notificaciones antes descripto.

En el momento que este fragmento de código es interpretado por el navegador web, primero crea la clase *AccelerationListener* y luego solicita al administrador de *listeners* (clase *ListenerManager*) para que agregue una instancia de la clase recién creada (linea 18). A partir de este momento una *SimpleCondition* puede registrarse para escuchar una "acceleration".

En el proceso de creación de la instancia de la clase *AccelerationListener*, por defecto se inicializa con una velocidad de refresco de *1000 ms* que la clase *Listener* se encargará de agregar al diccionario almacenado en el atributo *options* (Ver linea 3 de la Figura 4.10).

Luego, cuando se termine de cargar la página web, se le enviará un mensaje al método *startListening* para que termine de configurar el *timer* provisto por la interfaz de Phonegap (Ver linea 11 de la Figura 4.10).

```
1 AccelerationListener = Class.create(Listener, {
2   initialize: function($super) {
3     $super(1000);
4   },
5   sucess: function($super, listened) {
6     _.listeners.acceleration.changed(listened);
7     return true;
8   },
9   startListening: function() {
10    this.timerId = navigator.accelerometer.watchAcceleration(
11      this.sucess,
12      this.fail,
13      this.options
14    );
15  }
16 });
17
18 _.feel("acceleration", new AccelerationListener());
```

FIGURA 4.10: Código fuente de la clase AccelerationListener

Cuando la aplicación web se encuentre corriendo en el cliente, cada vez que el *timer* lo indique se llamará al método *success*. Este último, se encarga de enviar el mensaje *changed* a la instancia del listener para propagar la nueva lectura a todas las *SimpleCondition* que lo estén observando (Ver linea 6 de la Figura 4.10). A partir de esta propagación cada *SimpleCondition* evaluará si contiene al valor propagado por el listener utilizando el atributo *expression*.

Desde el punto de vista del desarrollador es necesario proporcionar una herramienta que simplifique la creación de condiciones. Para esto, en el servidor se implementa el patrón de diseño Builder explicado por *Gamma et al.*[2].

Luego, para cada sensor se puede proporcionar un Builder que contemple los casos específicos de dicho sensor. Por ejemplo, para el caso del acelerómetro, se crea la clase *OSCAccelerationBuilder* (ver Figura 4.11).

De esta forma, la clase *OSCAccelerationBuilder* tiene la responsabilidad de construir instancias de la clase *SimpleCondition* que serán atendidas por el listener que se encuentra del lado del cliente, registrado para atender condiciones de “acceleration”.

Por último, como se observa en el diagrama de clases UML (Ver Figura 4.11) la librería cuenta con la clase *OSCComplexConditionBuilder* que permite construir entornos complejos como consecuencia de agrupar un conjunto de condiciones.

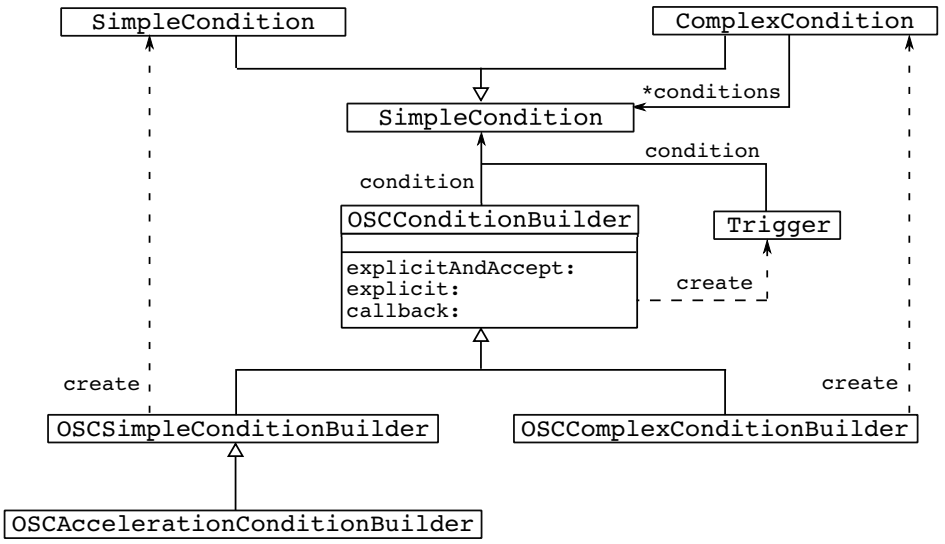

FIGURA 4.11: Diagrama de clases UML de la jerarquía de *builders*

4.7. Visión global

Con la misión de mantener, en la medida de lo posible, las características y propiedades de las aplicaciones web basadas en continuations, se diseña una extensión que permita la adaptación al contexto y mantienga la privacidad del usuario. Como se puede ver en el diagrama de clases *UML*[4] de la Figura 4.12, dicha extensión tiene dos partes, una se encuentra del lado del servidor y la otra del lado del cliente.

Para mantener dicha privacidad se plantea un esquema de notificaciones mediante el patrón de diseño *Publish/Suscribe* que evita que el cliente deba enviar información detallada de los sensores al servidor. Para lograrlo, cada notificación del cliente solicitará al servidor que realice la ejecución de un *callback de continuations* del framework *Seaside*, de forma similar a la forma de procesar clicks en hipervinculos o botones.

[4]Del inglés "Unified Modeling Language".

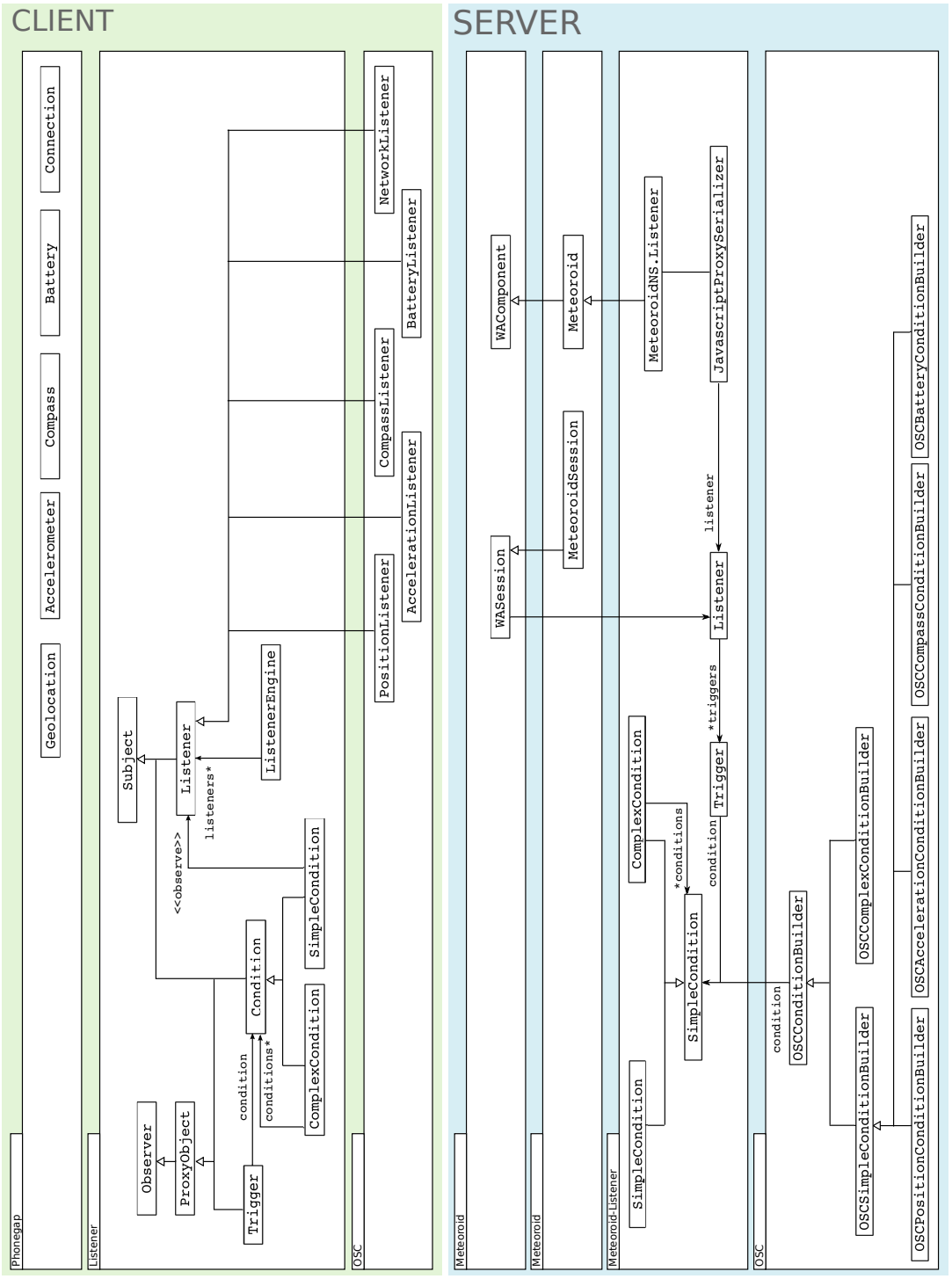

Figura 4.12: Diagrama de clases UML de la extensión propuesta

Capítulo 5

Caso de uso

Con el objetivo de poner a prueba la librería presentada, en este capítulo se describe como extender una aplicación web desarrollada con continuations para que pueda adaptarse al contexto.

Se utiliza como punto de partida una aplicación web que se encuentra en el repositorio publico de Cincom® dentro de los ejemplos de Seaside, e implementa una tienda *on line* que permite encargar sushi.

Antes de agregar la sensibilidad al contexto, se reemplazan el tipo de producto por otro que pueda ser identificado con un código de barra. En este caso, se utilizarán libros.

Además, se le realizan modificaciones a los componentes de Seaside para utilizar Scriptaculous y Meteoroid con el fin de reducir la transferencia de datos y mejorar la velocidad de refresco de cada actualización[25].

A continuación se detalla el modelo de negocio de la aplicación web previo a agregar la extensión. Luego se detallan los componentes de Seaside (junto con los de Scriptaculous y Meteoroid) que posibilitan la utilización del modelo. Para finalizar, una vez descripta la aplicación web, se explica como expandirla para proporcionar adaptabilidad al contexto.

5.1. El modelo de negocio de la aplicación web

El modelo de negocios de la tienda de libros consiste en registrar todas las ventas realizadas (clase Order). De cada venta se conoce el conjunto de libros que se solicitaron, la dirección de facturación y de entrega de los libros, y los datos de la tarjeta de crédito desde donde se realizó el pago (ver Figura 5.1).

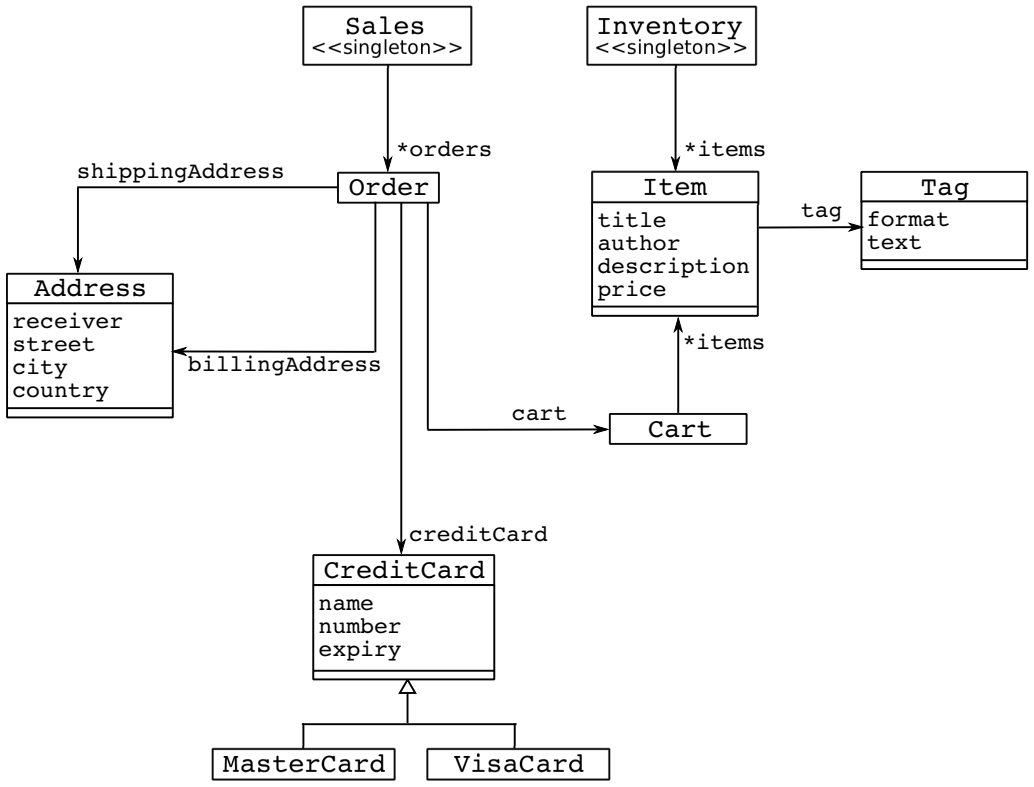

FIGURA 5.1: Diagrama de clases UML del modelo de negocio de la tienda de libros

De cada libro (clase *Item*) se conocen los atributos *titulo*, *autor/es*, *descripción*, *precio* y la *descripción de un código de barras* (clase *Tag*).

Las clases *Sales* y *Inventory* implementan el patrón de diseño *Singleton*[2], y se encargan de almacenar las compras consumadas y los productos del inventario, respectivamente.

5.2. La interfaz proporcionada por Seaside

Para poder interactuar con el modelo existen un conjunto de componentes de Seaside, que dan forma a la aplicación web accesible por el usuario.

Mientras el usuario navega por la tienda on line, realiza consultas sobre el catálogo de productos (clase *Inventory*, ver Figura 5.2). A su vez, puede agregar algún producto (clase *Item*) a un carro de compras (clase *Cart*), o sacarlo del carro de compras si es que ya no desea comprarlo (ver Figura 5.3).

Cuando el cliente desea hacer efectiva la compra, se le solicitan dos direcciones: una de envío del producto y otra para enviar la facturación (clase *Address*). Para cada dirección deberá completar los atributos *país*, *ciudad*, *calle* y *nombre de la persona que recibirá* ya sea los libros o la facturación.

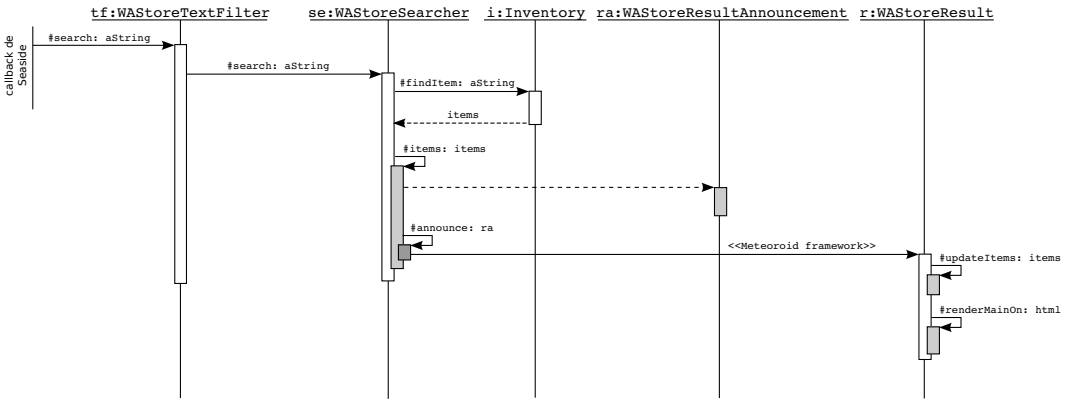

FIGURA 5.2: Diagrama de secuencia UML de la búsqueda de productos

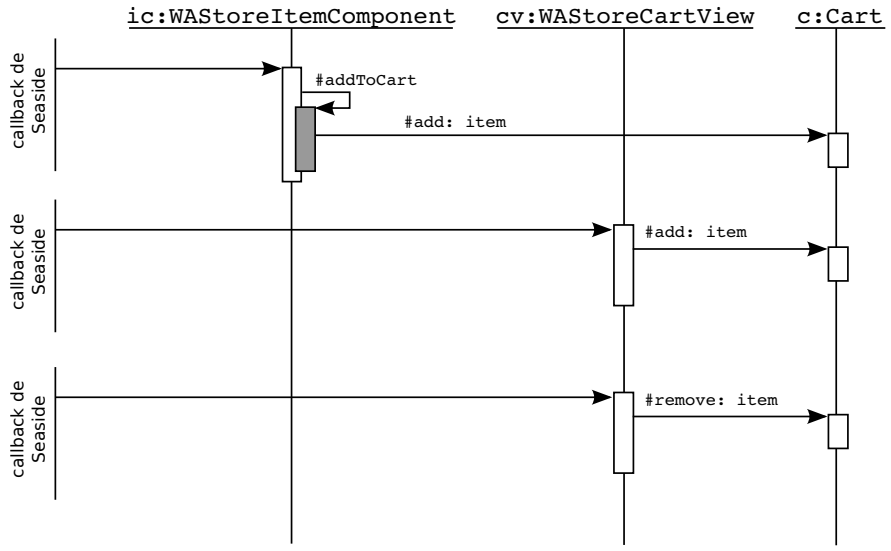

FIGURA 5.3: Diagrama de secuencia UML de la carga de productos al carro de compras

Por último, debe completar la información de su tarjeta de crédito (subclases de *CreditCard*) y confirmar la compra. La confirmación de la compra significa la creación de una instancia de la clase *Order* para registrar la compra. Esta instancia será automáticamente almacenada en la clase *Sales* (ver Figura 5.4).

Es importante notar que en el método *#go* de la clase WAStoreTask se detalla el proceso completo, desde que el cliente se encuentra modificando el carro de compras, hasta que termina de abonar por la compra realizada. En esta descripción, se aíslan 2 subprocesos, la *selección de los productos* y el *pago*.

Una vez que se *confirmó* la selección de los productos, el método *#isolate:* (perteneciente al framework Seaside) se encarga de descartar cualquier solicitud del navegador web que intente modificar el carro de compras. Lo mismo sucederá cuando se culmine de detallar la información de la tarjeta de crédito, bloqueando en este caso cualquier modificación de la orden generada.

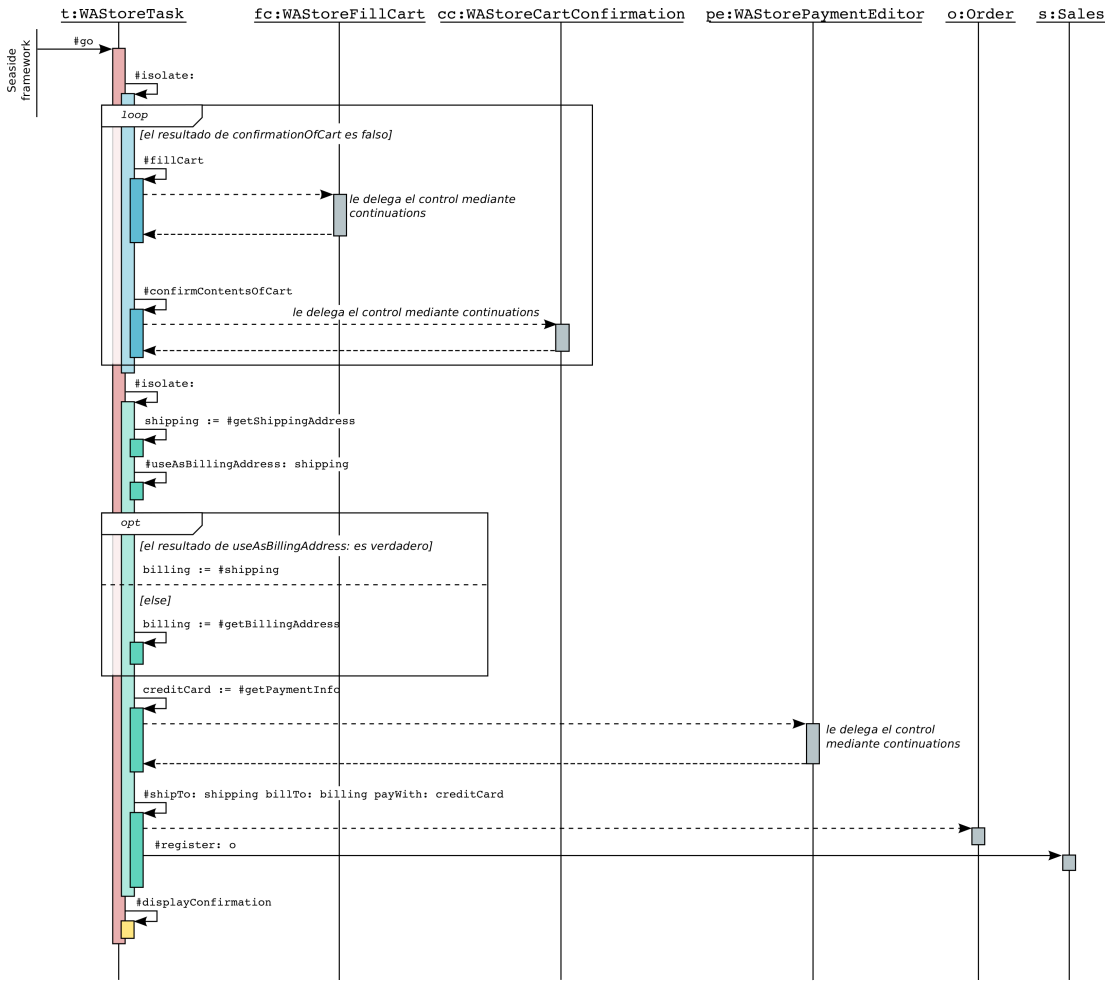

FIGURA 5.4: Diagrama de secuencia UML de la tienda de libros

5.3. ¿Cómo se utiliza la librería de adaptación al contexto?

Si se extendiese el ejemplo presentado mencionando que la tienda de libros posee varios locales de venta al público en los que desea proveer un servicio adicional cuando un usuario (que se encuentra utilizando la aplicación web) accede. El servicio en cuestión, consiste en simplificar la búsqueda de la descripción y el precio de un articulo, a partir de la utilización de un código de barra.

El primer paso para proporcionar la sensibilidad al contexto es que el componente raíz de la aplicación web extienda a la clase *MeteoroidNS.Meteoroid*.

Luego, es necesario agregar las librerías *PhonegapLibrary*, *ListenerEngineLibrary* y *OSCLibrary* a la aplicación web. Por último se deberá agregar una variable de instancia para almacenar el componente *MeteoroidNS.Listener*, que deberá ser *renderizado* al final del método *#renderContentOn: html* del componente raíz.

A continuación para cumplir con los nuevos requisitos de la tienda de libros, se detallará el uso del sensor de posicionamiento para detectar que el usuario se encuentre dentro de un local de venta. Luego, se explicará el acelerómetro, el cual es necesario para detectar que el usuario desea escanear un código de barras. Y por ultimo, un *sensor de códigos de barra* detectará si el código escaneado existe en el modelo de negocio.

5.3.1. Sensor de posicionamiento

Para detectar si el usuario se encuentra físicamente dentro de una tienda, es necesario establecer un polígono que delimite que posiciones pertenecen a una tienda. Luego, se utiliza a la librería para crear una *Condition* que comprobará si la posición actual de un dispositivo se encuentra dentro de dicho polígono[1]. Por último, es necesario asociar esta condición a un *trigger*, para que el servidor pueda coordinar la adaptación necesaria.

El código necesario para generar el trigger (ver Figura 5.5), primero crea a una instancia de la clase OSCPositionConditionBuilder al enviar el mensaje *#position* a la clase *OSCConditionBuilder*.

```
1  insideTrigger := (OSCConditionBuilder position)
2          inside: ((List new)
3                  add: -34.577301 @ -59.089359;
4                  add: -34.576801 @ -59.088829;
5                  add: -34.577690 @ -59.084999;
6                  add: -34.579041 @ -59.088242;
7                  add: -34.577793 @ -59.089119;
8                  add: -34.577339 @ -59.089279;
9                  add: -34.577301 @ -59.089359;
10                 yourself);
11         callback: [self whenInside].
```

FIGURA 5.5: Código fuente para comprobar si una posición se encuentra en una zona

Como se desea conocer cuando el cliente ingresa a una tienda, se envía el método *#inside:* a la instancia del builder con una lista de posiciones como parámetro. Estas coordenadas determinan el perímetro de uno de los locales en particular (ver Figura 5.6).

Luego, con el método *#callback:* se establece el comportamiento que debe realizar el servidor si el cliente *dispara* ese *trigger*. En este caso, enviará el mensaje *#whenInside* a la instancia que contenga la declaración anterior.

[1] Utilizando latitudes y longitudes como coordenadas.

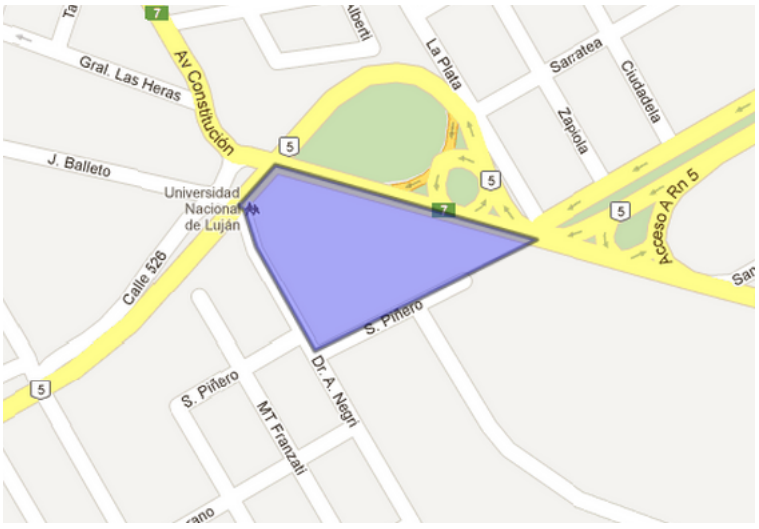

FIGURA 5.6: Mapa con el polígono en donde se encuentra la tienda

Dado que el trigger previo solo detecta cuando un usuario entra a una tienda, también es necesario controlar la condición opuesta para detectar cuando el cliente sale de la tienda, y en ese momento desactivar el servicio de búsquedas por código de barras

Lo único que será necesario cambiar será el método *#inside:* por *#outside:* (ver Figura 5.7).

```
outsideTrigger := (OSCConditionBuilder position)
        outside: ((List new)
                add: -34.577301 @ -59.089359;
                add: -34.576801 @ -59.088829;
                add: -34.577690 @ -59.084999;
                add: -34.579041 @ -59.088242;
                add: -34.577793 @ -59.089119;
                add: -34.577339 @ -59.089279;
                add: -34.577301 @ -59.089359;
                yourself);
        callback: [self whenOutside].
```

FIGURA 5.7: Código fuente para comprobar si una posición se encuentra fuera de una zona

A continuación, para que el servidor relacione a un trigger con un cliente en particular, es necesario solicitarle a una instancia de la clase *Listener* (accesible desde la sesión de usuario de Seaside[2]) que lo *recuerde* mediante el método *#remember:* (ver Figura 5.8).

Por el contrario, para que el navegador web deje de ser sensible a un *trigger* deberá indicarle al *listener* que lo *olvide* utilizando el método *#forget:* (ver Figura 5.9).

[2]Es importante destacar que todo el comportamiento de la sensibilidad al contexto se describe dentro de los *WAComponent* de Seaside, y fuera del modelo de negocio.

```
1 self session listener remember: insideTrigger.
```

FIGURA 5.8: Código fuente para que un cliente sea sensible a un trigger.

```
1 self session listener forget: insideTrigger.
```

FIGURA 5.9: Código fuente para que un cliente deje de ser sensible a un trigger.

5.3.2. Sensor de aceleración

Luego de detectar la posición del usuario, si este se encuentra en una tienda es necesario reconocer si su intención es escanear un código de barras.

Para lograr esto, cuando se encuentra dentro de una tienda (el método *#whenInside*), se deberán registrar que posturas del dispositivo activarán la lectura de códigos de barra y en que posturas estas deberían desactivarse.

Por ejemplo, cuando el dispositivo se encuentra de forma vertical se asume que el usuario se encuentra en una postura ideal para utilizar la aplicación web. Por el contrario, cuando el usuario posiciona el dispositivo en forma apaisada, será interpretado como que el usuario desea leer un código de barras (ver Figura 5.10).

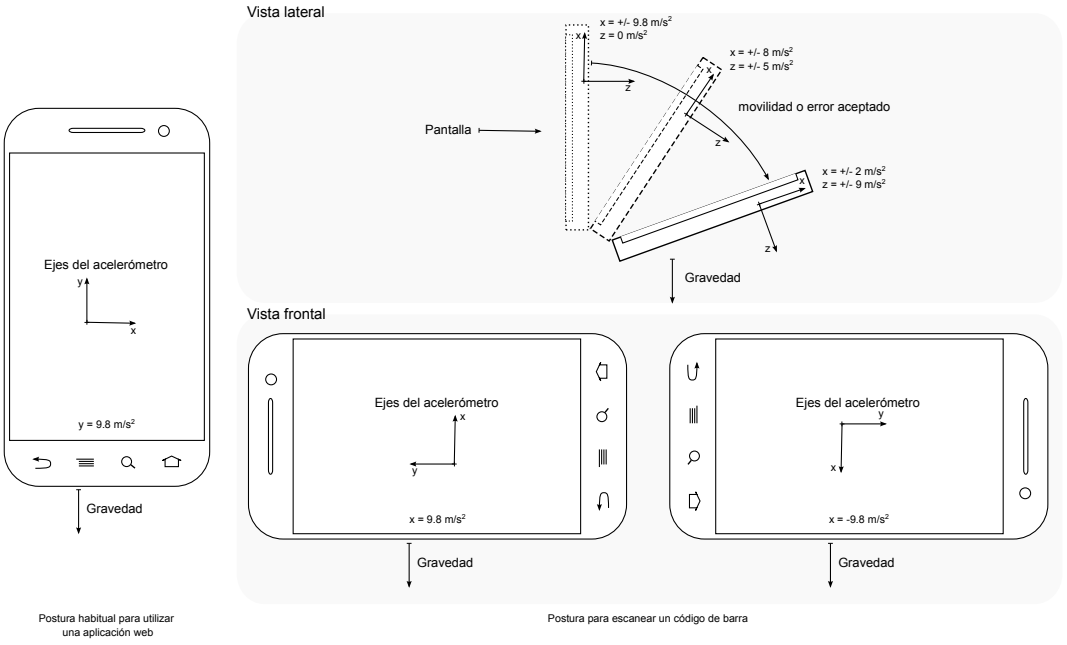

FIGURA 5.10: Postura vertical y horizontal de un dispositivo

Al describir la postura de escaneo de códigos de barra se tiene en cuenta cierta movilidad o margen de error mostrado en la *vista lateral*. Esta flexibilidad permitirá rotar hacia atrás el dispositivo cuando se encuentra en horizontal.

Luego, para definir la postura de escaneo es necesario utilizar 2 triggers, uno para cada tipo de apaisamiento. El primer trigger se encargará de detectar un angulo de 90 grados desde la posición vertical en sentido contrario a las agujas del reloj, mientras que segundo trigger se encargará de los 90 grados en sentido horario.

Teniendo en cuenta los extremos para cada una de las posturas se plantea como valores medios los puntos [x:7, y:0, z:5] y [x:-7, y:0, z:5] (ver Figura 5.11). Luego el margen de error es para ambas posturas iguales [x:5, y:1, z:5][3]

```
scanningPosture1 := (OSCConditionBuilder acceleration)
        when: 7 @ (0 @ 5);
        acceptError: 5 @ (1 @ 5);
        callback: [self wantScanBarcodes].
scanningPosture2 := (OSCConditionBuilder acceleration)
        when: -7 @ (0 @ 5);
        acceptError: 5 @ (1 @ 5);
        callback: [self wantScanBarcodes].
```

FIGURA 5.11: Código fuente para describir la postura de escaneo de código de barras

Nuevamente, es necesario recordar que habrá que informarle al *Listener* de la sesión del usuario mediante el método *#remember:* si es que deseamos reconocer este entorno.

Por otra parte, se determina que el usuario quiere navegar por la aplicación web cuando no se encuentra en una postura de escaneo. Para identificar esto se define un trigger con una condición compleja (ver Figura 5.12).

Dentro del bloque de código que se le pasa como parámetro al método *#when:*, la variable *env* permitirá acceder a la clase *OSCConditionBuilder* dentro de la condición compleja. También es importante notar el método *#whenNot:* que niega la ocurrencia de esa condición. Por último, el operador & (comúnmente llamado "and") permite la concatenación de condiciones para formar una más compleja.

[3] Al eje del acelrómetro y se le deja un margen de error de ± 1 m/s^2 para que el usuario pueda encontrar de forma sencilla la postura para escanear el código de barra, y no tenga que lidiar con la sensibilidad ofrecida por los acelerómetros de distintos dispositivos.

```
navigationPosture := (OSCConditionBuilder complex)
        when: [:env |
                ((env acceleration)
                        whenNot: -7 @ (0 @ 5);
                        acceptError: 5 @ (1 @ 5))
                &
                ((env acceleration)
                        whenNot: 7 @ (0 @ 5);
                        acceptError: 5 @ (1 @ 5))];
        callback: [self wantNavigate].
```

FIGURA 5.12: Código fuente para describir la postura de escaneo de código de barras

5.3.3. Sensor de códigos de barra

En el momento que el *acelerómetro* detecte que el usuario se encuentra en la postura adecuada para escanear un código de barra, la aplicación web deberá cargar en el dispositivo móvil del cliente los triggers adecuados para reconocer el código de barra de cada libro (ver Figura 5.13).

```
bookTriggers := OrderedCollection new.
Inventory default allItems do: [:item |
        bookTriggers add: ((OSCConditionBuilder tag)
                when: item tag format | item tag text;
                callback: [ self findTaggedWith: item tag ])]
```

FIGURA 5.13: Código fuente para describir las condiciones relacionadas con códigos de barras aceptados por el modelo

En este caso se agregan todos los triggers a una colección de objetos, notar que en el método *#when:* se envía como parámetro el *formato* del código y el *texto* que simboliza la secuencia de caracteres o dígitos del código.

Para finalizar será necesario solicitarle al *Listener* de la sesión del usuario que recuerde cada uno de los triggers contenidos en la colección *bookTriggers*, para que el navegador pueda utilizarlos para reconocer entornos[4].

Como consecuencia del proceso anterior, si el cliente enfoca la cámara de su dispositivo hacia el código de barras de un libro, el navegador web procederá a mostrar la información relacionada con dicho libro.

Cabe señalar que así como se describieron estos sensores, necesarios para desarrollar el caso de uso planteado, existen otros (como el estado de la batería, el tipo de conexión

[4]Del lado del cliente, el navegador solo mostrará la pantalla de escaneo de códigos de barras si es que existe alguna condición dentro del *listener* de códigos de barras.

a internet, la orientación con respecto al Polo Norte, etc.) que también pueden ser utilizados. Además, si la situación lo requiere, se pueden crear una gran cantidad de nuevos sensores que ayuden a reconocer las necesidades del usuario.

Capítulo 6

Extensibilidad y límites

Como consecuencia de la librería presentada se desprenden posibles desarrollos y extensiones que podrían mejorar la adaptabilidad al contexto, o por otra parte, la privacidad y la seguridad del usuario.

Por otra parte, dado que al realizar la librería se han tomado ciertas decisiones de diseño, habrá limites que no podrán ser solucionados mediante las extensiones. Por esto, es necesario describir cuales serán los límites de la librería con motivo de determinar el alcance de la solución planteada.

A continuación se presentan algunas de las extensiones posibles relacionadas con las distintas lineas de investigación en las que se sostiene esta tesis. Por último se prosigue con la descripción de los limites encontrados y se mencionan que características de la sensibilidad al contexto no podrán utilizarse si se mantiene lo propuesto en esta tesis.

6.1. Extensibilidad

Al momento de desarrollar la librería surgieron diferentes ideas que son las que le dieron forma, pero también existieron otras opciones que fueron pospuestas en pos de acotar el alcance de la tesis.

Estas posibles extensiones pueden ser clasificadas en: las que agregan nuevos *sensores* al navegador web mediante *plugins*; y las que mejoran la privacidad y seguridad del usuario.

En el primer grupo, se encuentran aquellos nuevos sensores que permitirán reconocer de forma mas precisa el contexto. Estos sensores podrán reconocer distintas características

del contexto como la temperatura, la humedad, la presión atmosférica, u otros entornos aún no pensados.

Dentro de este mismo grupo, también se pueden crear sensores que notifiquen a cada aplicación web los recursos con los que cuenta para realizar su ejecución, y así proporcionarle mayor información sobre las capacidades que el dispositivo está dispuesto a proporcionarle en un momento particular. Por ejemplo, notificarle cuando se le brinda mas capacidad de procesamiento o cuando se le quita dicho recurso.

Dicha información permitirá establecer requisitos mínimos sobre ciertos recursos del dispositivo móvil, permitiendo descartar ciertas adaptaciones si es que el dispositivo no es capaz de procesarlas de forma adecuada.

A su vez, al profundizar este último caso, es necesario contemplar la ejecución en paralelo de distintas aplicaciones web e identificar la necesidad de un método que permita su *interacción* (semejante a lo planteado por *Efstratiou*[14, p.˜5]).

En el segundo grupo, se encuentran aquellas ideas que modifiquen la administración de privacidad relacionada con los sensores, mediante una mayor flexibilidad de la interfaz gráfica o a través de la simplificación la interacción con el usuario.

Existen varias alternativas de investigación dentro de este grupo. Primero, sería interesante evaluar si la utilización del patrón de diseño *Composite*[2, p.˜151] mejora la interfaz de administración de permisos. Luego, queda pendiente verificar si la utilización de *permisos asociativos*[1] simplifica la administración de accesos o solo le quita transparencia al proceso. Por último, quedaría corroborar como afectan estas nuevas estrategias de permisos y accesos al *Same Origin Policy (SOP)*[2] presente en los navegadores web.

Dentro de ésta segunda agrupación, cada modificación puede ser evaluada teniendo en cuenta la usabilidad que le proporciona al usuario y la maleabilidad para establecer permisos entre las distintas aplicaciones web.

6.1.1. Interacción entre aplicaciones web

En el uso cotidiano de un navegador web es habitual la carga y ejecución en simultaneo de varias aplicaciones web, aunque se visualice solo una de ellas. Estas ejecuciones concurrentes de aplicaciones, combinadas de forma inadecuada con la adaptación al contexto, puede producir los inconvenientes ya planteados por *Efstratiou*[14, p.˜5].

[1] El *permiso asociativo* consiste en definir el permiso de una aplicación A a partir de los permisos concedidos a otras aplicaciones B y C.

[2] La *Same Origin Policy* es una regla que establece que un *script* de Javascript solo se pueda acceder a los métodos y propiedades asociados a su origen, restringiendo el intercambio de información entre scripts de distintos orígenes.

En particular, *Efstratiou* presenta un escenario en donde existen una aplicación sensible al consumo energético y otra que es sensible al ancho de banda de una red inalámbrica (por ejemplo un reproductor de música de una radio online).

En un punto de la ejecución, la primer aplicación reduce el consumo de ancho de banda de la red con el fin de reducir el consumo energético. Al instante sucesivo, la segunda aplicación encuentra que puede aumentar la calidad del audio dado que el sistema dispone de un mayor ancho de banda recientemente libreado por la primer aplicación.

Como resultado de estas adaptaciones independientes, las acciones de adaptación de la primera aplicación serán canceladas por las adaptaciones realizadas de la segunda. Para sobrellevar esto *Efstratiou* plantea una solución en conjunto en donde se contemplan los requisitos y las prioridades de cada aplicación [14, p.˜58]. En esta linea queda pendiente analizar la solución teniendo en cuenta la utilización de continuations y los requisitos de seguridad existentes en los navegadores web.

Esta última linea, a su vez, se puede continuar mediante el análisis de la capacidad de intercomunicar dos aplicaciones web mediante un canal proporcionado por el navegador web.

Un canal de tales características podría mejorar el rendimiento de ambas aplicaciones al proveer un mecanismo de sincronización, este mecanismo podría permitir la colaboración entre las distintas aplicaciones web y por consecuencia evitar el procesamiento por duplicado.

En este tipo de comunicación entre aplicaciones, también debería contemplarse algún tipo de control para mantener la seguridad de la aplicación.

Administración de permisos mediante Composite

Teniendo en cuenta que el diseño presentado promociona la creación de una gran cantidad de sensores, en un futuro surgirá la necesidad de mejorar la administración de accesos para que un usuario sea capaz de discernir y decidir que información desea compartir con cada una de las aplicaciones web.

El módulo de administración tendrá que considerar que cada aplicación web puede requerir una configuración distinta de accesos, y deberá evaluar cada cuando tiempo un usuario modifica los permisos ya concedidos para una aplicación web particular.

El patrón de diseño Composite puede ser una estructura válida para proveer una gran granularidad en el control de los permisos y aún así, en caso de necesidad, permitir la

abstracción mediante la agrupación de alguno de estos permisos; ya sean estos simples o consecuencia de otra agrupación previa.

En este planteo se deberá investigar si existen grupos de permisos que se repiten en diferentes aplicaciones. Y, por otra parte, si la agrupación en si misma simplifica la tarea de configuración de permisos que suele llevar a cabo el usuario.

Permisos asociativos

Ya sea como complemento del caso anterior, o como investigación individual, puede llegar a tener cierta connotación la posibilidad de determinar los permisos de una aplicación web a partir de la unión de permisos de una o mas aplicaciones web.

En este tipo de estrategia de administración el foco se centra en establecer los permisos a partir de semejanzas con otras aplicaciones web. Dada una aplicación web que fue desarrollada por una organización, una segunda aplicación web de esta organización recibiría como mínimo los mismos permisos que la primera aplicación desarrollada.

Y, mediante este enfoque, el usuario podría establecer explícitamente desde una interfaz gráfica que los permisos de la segunda aplicación web son idénticos a los de la primera. De esta forma, si los permisos concedidos a la primera se modificasen, la segunda aplicación web automáticamente conocería las mismas restricciones.

En esta linea se deberá analizar si el usuario es capaz de discernir en todo momento los permisos que se encuentra administrando.

Estrategias para extender el Same Origin Policy

Aunque en el ejemplo considerado en el capítulo 5 toda la aplicación se descarga desde el mismo *origen*[3], es posible que en otros casos la aplicación web se descargue desde múltiples *orígenes*. Teniendo en cuenta dicha situación, es necesario evaluar una solución que permita conceder distintos niveles de permisos para cada origen.

Dos posibles soluciones a este problema podrían ser: permitir solo al origen principal acceder a la información de los sensores, o distinguir orígenes secundarios y permitir una administración de permisos mas detallada.

En la primer solución, será necesario realizar la extensión correspondiente a la Same Origin Policy bloqueando cualquier tipo de acceso a los orígenes secundarios.

[3] Un *orígen* se encuentra definido por el *dominio*, *puerto* y el *protocolo* utilizados en una *URL* particular.

Por otra parte, para la segunda solución, deberán considerarse alternativas mas dinámicas de administración de permisos que simplifiquen y clarifiquen las selecciones realizadas por el usuario en cuanto a la información que desea compartir. En este punto volverían a tener vigencia tanto los *permisos asociativos*, como los *permisos mediante Composite*.

6.2. Limites

Al considerar la flexibilidad proporcionada por la creación de nuevos sensores, también es necesario analizar cuales serán los limites de este diseño.

La base del diseño consiste en que la aplicación web proporcione una mayor privacidad al usuario al utilizar *triggers* y notificar al servidor solo en caso de que un *entorno* se cumpla.

Al proteger la información de los sensores manteniéndola del lado del cliente y solo notificar lo indispensable al servidor, este último nunca recibirá información como para realizar un análisis complejo que le permita detectar nuevos patrones de comportamiento del cliente.

Esta situación es la que determina que el servidor nunca podrá implementar la característica evolutiva de la sensibilidad al contexto.

Capítulo 7

Conclusión

Como consecuencia del diseño y el desarrollo realizados, ha quedado demostrada la existencia de una alternativa que permita a una aplicación web desarrollada con continuations utilizar los sensores de un dispositivo.

Teniendo en cuenta los objetivos de esta tesis y las características de las posibles extensiones, y habiendo contemplando a su vez los límites de la idea presentada, a continuación se realizará una comparación con otros trabajos de semejantes características.

Luego se enumerarán los resultados obtenidos a partir del diseño propuesto. Y por último se presentarán algunos de las trabajos pendientes que requieren de un análisis mas profundo.

7.1. Trabajos relacionados

En la introducción a este documento se mencionaron otros trabajos que mantienen relación con alguna de las lineas de investigación desarrolladas. Para distinguir el aporte realizado por este trabajo se procede con la descripción de las principales diferencias entre el diseño presentado y el enfoque presentado por el resto de los trabajos.

Al contemplar el aporte realizado por *Queinnec*[12], que resuelve los inconvenientes relacionados con las transacciones y los estados de una aplicación web mediante la utilización de Continuations, este diseño no hace mas que agregar una extensión que permita el manejo de información contextual a este tipo de aplicaciones.

Luego, siguiendo por la linea de las aplicaciones web se encuentra el trabajo de *Challiol et al.*[3], que ha sido desarrollado con MVC. En este trabajo se implementa un Controlador que es el encargado de manejar la información contextual, aunque esta solo consista

en información de localización, que puede ser obtenida por múltiples sensores que se encuentran distribuidos en el ambiente.

A diferencia de *Challiol et al.*, en este trabajo se presentan ejemplos en los que se restringe la ubicuidad de los sensores y se determina que la información es siempre proporcionada por el dispositivo móvil del cliente, con el fin que el usuario siempre tenga el control de su privacidad. Además los valores leídos por los sensores nunca son enviados a los servidores, con el fin de mejorar la privacidad y seguridad del usuario.

Por otra parte *Chang et al.*[16] plantea un mecanismo de adaptación web que se enfoca en incorporar la sensibilidad al contexto contemplando principalmente la QoS. Su enfoque consiste en decidir de que lado de la conexión conviene ejecutar una serie de bloques (o componentes). Este tipo de adaptación puede ser realizada solo en el momento que la aplicación web se carga en el cliente, perdiendo la aplicación cualquier posibilidad de realizar nuevas adaptaciones en tiempo de ejecución.

A diferencia de *Chang et al.*, en este trabajo se provee mayor flexibilidad para decidir el momento de aplicar una adaptación. Por otro lado, *Chang et al.* contempla la variante evolutiva al momento de diseñar su modelo.

En el 2009, *Kapitsaki et al.*[6] define que una forma de permitir la sensibilidad al contexto es mediante la composición de un servicio o aplicación web con un servicio de sensibilidad al contexto, mediante la utilización de otro servicio web que se encarga de la composición. Esto implica que la sensibilidad al contexto queda definida fuera del modelo de negocios.

Parte de estas nociones, proporcionadas por *Kapitsaki et al.*, han sido utilizadas al momento de definir como se relaciona la sensibilidad al contexto con el modelo de negocio. A diferencia de *Kapitsaki et al.*, en este trabajo, la composición entre la sensibilidad al contexto y el modelo de negocio ocurren dentro de los componentes de Seaside.

El resto de los trabajos (*Efstratiou*[14] y *Fortier et al.*[7]) se encuentran relacionados con la sensibilidad al contexto y proponen el desarrollo de aplicaciones que no corren en navegadores web, por lo que la privacidad del usuario nunca surgió como un tópico a analizar.

El trabajo de *Efstratiou*[14] consiste en desarrollar un procedimiento basado en pólizas que permite administrar de forma coordinada las adaptaciones sugeridas por múltiples aplicaciones. Estas pólizas consisten en disparar *triggers* para llevar a cabo las adaptaciones necesarias.

Aunque en el diseño presentado en el capítulo 4 se utiliza el concepto de triggers de *Efstratiou* y se mencionan posibles extensiones que produzcan una adaptación coordinada,

fue necesario adaptar lo propuesto por *Efstratiou* a una arquitectura de aplicaciones web basada en Continuations.

En comparación con la propuesta de *Costanza*[15], que presenta el *COP* como un nuevo paradigma, esta librería tiene un alcance mas limitado que es brindar sensibilidad al contexto a aplicaciones web desarrolladas con continuations.

Por último *Fortier et al.*[7], plantea una plataforma para tratar con la complejidad de un software sensible al contexto. Esta plataforma está compuesta por 4 partes: el *soporte de sensibilidad*, las *adaptaciones específicas del dominio*, un *adaptador a aplicaciones existentes* y por último el módulo de conexión de las tres partes anteriores compuesto por *abstracciones centrales*.

La principal diferencia con el diseño de *Fortier et al.* consiste en el punto de conexión entre el modelo contextual y el modelo de negocio. *Fortier et al.* considera que el modelo contextual se encuentra directamente relacionado con el modelo de negocios, de esta forma se torna mas difícil que cada interfaz de usuario establezca nuevas formas de comunicar ambos modelos (el contextual y el de negocios).

Para sobrellevar dicha situación, en este documento se presentó un lenguaje que permite definir la relación entre el modelo contextual y el modelo de negocios dentro de una subclase de Component (que se utilizan para definir interfaces en Seaside).

7.2. Lecciones aprendidas

Al concluir este trabajo se ha logrado cumplir con tres los objetivos que lo motivaron: El desarrollo, la explicación mediante patrónes de diseño y la comparación con otras alternativas.

En primer lugar, se ha proporcionado un mecanismo para que una aplicación web desarrollada con continuations se pueda adaptar al contexto del usuario. Esta adaptación permitirá que aplicaciones web que en la actualidad se encuentran realizadas con continuations puedan a su vez adaptarse al contexto.

Luego, se han explicado las decisiones de diseño mediante la utilización de patrónes de diseño como: *Adapter*, *Composite*, *Publish/Subscriber* y *Builder*.

El *Adapter* ha permitido estandarizar las interfaces de los posibles sensores para que la utilización de sus valores pueda realizarse mediante condiciones preestablecidas.

Luego, el *Composite* ha proporcionado flexibilidad y simplicidad para combinar distintas condiciones, permitiendo a su vez realizar ciertas optimizaciones en el procesamiento.

Posteriormente, el *Publish/Subscriber* ha otorgado una forma de mantener la privacidad del usuario, además de reducir el uso de la red en comparación con otras alternativas basadas en MVC.

El *Builder*, por otra parte, ha simplificado la forma de definir condiciones, permitiendo que el desarrollador siga enfocado el la lógica del negocio, y ahora con la posibilidad de aprovechar la información contextual.

Por último, en la sección Trabajos relacionados (7.1) se ha realizado una breve mención de las diferencias existentes con otras alternativas.

7.3. Trabajo futuro

Luego de la conclusión del objetivo planteado en este documento, y considerando las distintas alternativas para continuar esta nueva linea de investigación[1], en un futuro me encontraré analizando el rendimiento, en términos de necesidad computacional y de comunicación de la sensibilidad al contexto, entre las aplicaciones web basadas en continuations y aquellas que utilizan el esquema MVC.

En esta linea, también ha quedado pendiente mejorar los Builders de condiciones tanto simples como complejas de forma tal que utilicen todas las posibilidades de la *lógica difusa* para definir los diferentes entornos.

Por otra parte, dentro del navegador web, es necesario analizar la definición de una API que estandarice la definición de sensores. De esta forma reducirá la necesidad de definir el comportamiento específico de cada *Listener* del lado de *javascript*.

Además, analizaré que posibilidades hay de definir una Ontología que permita la utilización de nuevos sensores sin la necesidad de modificación del navegador web. Esto permitirá agregar nuevos sensores, sin la necesidad de volver a compilar el navegador web. En esta linea, tal vez sea posible detectar de forma automática que sensores se encuentran disponibles en un dispositivo, y ponerlos a disposición para que el usuario pueda utilizarlos.

En otro aspecto, es necesario mejorar e investigar una alternativa que mejore la percepción del usuario en torno a la administración del acceso y la información privada. En esta etapa, será necesario realizar un análisis de usabilidad de la interfaz de configuración y ofrecer una alternativa que sea simple e intuitiva.

[1] Considerando que es necesaria una linea de investigación que se enfoque en analizar a las aplicaciones web basadas en continuations en combinación con la sensibilidad al contexto.

Por último, continuando con la administración del acceso, también me enfocaré en analizar la *interacción entre aplicaciones web*, la *administración de permisos mediante Composite*, los *permisos asociativos* y posibles *estrategias para extender el Same Origin Policy* (Ver el capítulo 6, Extensibilidad y límites).

Bibliografía

[1] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *JOOP*, August 1988.

[2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, October 1995. ISBN 0-201-63361-2. URL e-stud.vgtu.lt/users/files/dest/1542/designpatterns.pdf.

[3] Cecilia Challiol and Valeria Soledad De Cristófolo. Extensión del MVC para resolver problemas de Hipermedia Física. Master's thesis, Universidad Nacional de La Plata, Argentina, 2006.

[4] Andrés Fortier, Cecilia Challiol, Gustavo Rossi, and Silvia Gordillo. Physical hypermedia: a context-aware approach. *Springer Verlag LNCS*, June 2007. URL http://www.lifia.info.unlp.edu.ar/papers/2007/Fortier2007.pdf.

[5] Cecilia Challiol, Andrés Fortier, Silvia Gordillo, and Gustavo Rossi. Architectural and implementation issues for a context-aware hypermedia platform. 2008. URL http://www.lifia.info.unlp.edu.ar/papers/2008/Cecilia2008.pdf.

[6] Georgia M. Kapitsaki, Dimitrios A. Kateros, George N. Prezerakos, and Iakovos S. Venieris. Model-driven development of composite context-aware web applcations. *Information and Software Technology*, 51:1244–1260, March 2009. URL http://s2e.teipir.gr/papers/Model-driven_development_of_composite_context-aware_web_applications.pdf.

[7] Andrés Fortier, Gustavo Rossi, Silvia E. Gordillo, and Cecilia Challiol. Dealing with variability in context-aware mobile software. *The Journal of Systems and Software*, 83:915–936, November 2009. URL http://se.ce.pusan.ac.kr/xe/?module=file&act=procFileDownload&file_srl=26172&sid=78ff8767630c94766db93f350a0f0dd9.

[8] Bill Schilit and Marvin Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 0890-8044/94:22–32, September/October 1994. URL http://impact.asu.edu/~cse591uc/papers/00313011.pdf.

[9] John C. Reynolds. The Discoveries of Continuations. *LISP and Symbolic Computation: An international Journal*, 6:233–247, 1993. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.4705&rep=rep1&type=pdf.

[10] Adriaan van Wijngaarden. Recursive definition of syntax and semantics. *Formal Language Description Languages for Computer Programming*, pages 13–24, 1966. URL http://oai.cwi.nl/oai/asset/9205/9205A.pdf.

[11] Jonathan Rees, H. Abelson William Clinger, N. I. Adams IV, D. H. Bartley, G. Brooks, R. K. Dybvig, D. P. Friedman, R. Halstead, C. Hanson, C. T. Haynes, E. Kohlbecker, D. Oxley K. M. Pitman, G. J. Rozas, G. J. Sussman, and M. Wand. Revised(3) Report on the Algorithmic Language Scheme. Technical report, 1988. URL http://groups.csail.mit.edu/mac/ftpdir/scheme-reports/r3rs-html/r3rs_toc.html.

[12] Christian Queinnec. Inverting back the inversion of control or, continuations versus page-centric programming. May 2001. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.2753&rep=rep1&type=pdf.

[13] Anind K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5:4–7, 2001. URL http://www.cc.gatech.edu/fce/ctk/pubs/PeTe5-1.pdf.

[14] Christos Efstratiou. Coordinated Adaption for Adaptive Context-Aware Applications. Master's thesis, Lancaster University, United Kingdom, May 2004. URL http://eprints.lancs.ac.uk/12455/1/EfstratiouThesis.pdf.

[15] Pascal Costanza. Context-oriented Programming in ContextL - State of the Art. *ACM 2008*, 2008. URL http://p-cos.net/documents/contextl-soa.pdf.

[16] Po-Hao Chang and Gul Agha. Towards Context-Aware Web Applications. *International Federation for Information Processing*, pages 239–252, 2007.

[17] David A. Ladd and J. Christopher Ramming. Programming the web: An application-oriented language for hypermedia service programming. In *4th International World Wide Web Conference - WWW4*, pages 567–586. World Wide Web Consortium, December 1995. URL http://www.w3.org/Conferences/WWW4/Papers/251/.

[18] Stéphane Ducasse, Adrian Lienhard, and Lukas Renggli. Seaside - a multiple control flow web application framework. In *ESUG Conference Research Track*, pages 231–254, 2004. URL http://scg.unibe.ch/archive/papers/Duca04eSeaside.pdf.

[19] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. URL http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[20] Paul Dourish. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8, 2004. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.8277&rep=rep1&type=pdf.

[21] Shanwei Cen. *A Software Feedback Toolkit and its Application in Adaptive Multimedia Systems*. PhD thesis, Tsinghua University, China, October 1997.

[22] Mieczyslaw M. Kokar, Kenneth Badawski, and Yonet A. Eracar. Control theory-based foundations of self-controlling software. *Intelligent Systems and their Applications, IEEE*, 14(3):37–45, June 1999.

[23] Alex Meng. On evaluating self-adaptive software. In Paul Robertson, Howie Shrobe, and Robert Laddaga, editors, *Self-Adaptive Software*, volume 1936 of *Lecture Notes in Computer Science*, pages 65–74. Springer Berlin / Heidelberg, 2001. ISBN 978-3-540-41655-5.

[24] Kenneth P. Birman and Thomas A. Joseph. Exploiting virtual synchrony in distributed systems. In *SOSP '87 Proceedings of the eleventh ACM Symposium on Operating systems principles*, 1987. URL www.cs.cornell.edu/home/rvr/sys/p123-birman.pdf.

[25] Juan Lautaro Fernández, Santiago Robles, Andrés Fortier, Stéphane Ducasse, Gustavo Rossi, and Silvia Gordillo. Meteoroid - towards a real mvc for the web. In *IWST'09*, August 2009. URL http://rmod.lille.inria.fr/archives/workshops/Laut09a-IWST09-Meteroid.pdf.